VICON
MOTION SYSTEMS

© 2004–2005 Vicon Motion Systems Limited. All rights reserved.

Vicon Motion Systems Limited reserves the right to make changes to information in this document without notice. Companies, names, and data used in examples are fictitious unless otherwise noted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic or mechanical, by photocopying or recording, or otherwise without the prior written permission of Vicon Motion Systems Limited.

Information furnished by Vicon Motion Systems Limited is believed to be accurate and reliable; however, no responsibility is assumed by Vicon Motion Systems Limited for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of Vicon Motion Systems Limited.

Vicon® is a registered trademark of OMG Plc. PECS™ and Workstation™ are trademarks of OMG Plc.

Other product and company names herein may be the trademarks of their respective owners.

California
9 Spectrum Pointe
Lake Forest
CA 92630
USA

Tel: +1 (949) 472 9140
Fax: +1 (949) 472 9136

Colorado
7388 S. Revere Parkway, Suite 901
Centennial
CO 80112
USA

Tel: +1 (303) 799 8686
Fax: +1 (303) 799 8690

UK
14 Minns Business Park
West Way
Oxford OX2 0JB
UK

Tel: +44 (0)1865 261800
Fax: +44 (0)1865 240527

Vicon Motion Systems is an OMG Plc company
Email: support@vicon.com
Web: http://www.vicon.com

1 INTRODUCTION AND REQUIREMENTS 9

2 INSTALLING THE PECS SERVER 10
2.1 Installation and Registration 10
2.2 Installing the Server in Client Applications 10
2.2.1 Microsoft Office Products 10
2.2.2 Visual Basic 11
2.2.3 Microsoft Visual C++ 12
2.2.4 Microsoft Visual J++ 12
2.2.5 MATLAB 13
2.2.6 LabVIEW 13

3 USING THE AUTOMATION SERVER 14
3.1 Plug-In Options 15
3.1.1 Using Options Files 15
3.1.2 Managing the Pipeline 16
3.1.3 Adding Processes to the Pipeline 16
3.1.4 Process Options 17
3.1.5 Debugging Client Code 19
3.1.6 Settings 20
3.1.7 Product Version Information 21
3.1.8 Releasing Client Applications 21
3.2 Server Options 22
3.3 Writing Client Processes using the ActiveX Interface 24
3.3.1 The 'DUAL' ActiveX Interface 25
3.3.2 Features not for External Use 25
3.4 Sample Code 26
3.4.1 Visual Basic (and VBA) 26
3.4.2 MATLAB 26
3.4.3 LabView 27

4 ALPHABETIC INTERFACE REFERENCE 28
4.1 AnalogChannel 28
4.1.1 AnalogTo VideoRatio 28
4.1.2 BaseRateDivider 29
4.1.3 FirstSampleNum 29
4.1.4 GetSample 29
4.1.5 GetSamples 29
4.1.6 LastSampleNum 29
4.1.7 SampleRate 30
4.1.8 SetSample 30
4.1.9 SetSamples 30
4.1.10 Units 30

4.1.11	MatchesLabel	30
4.1.12	Label	31
4.2	EclipseNode	32
4.2.1	DataPath	32
4.2.2	NodeID	32
4.2.3	Title	33
4.2.4	GetBoolean Attribute	33
4.2.5	GetDateAnd TimeAttribute	33
4.2.6	GetInteger Attribute	33
4.2.7	GetRealAttribute	34
4.2.8	GetTextAttribute	34
4.2.9	SetBoolean Attribute	34
4.2.10	SetDateAnd TimeAttribute	34
4.2.11	SetInteger Attribute	35
4.2.12	SetRealAttribute	35
4.2.13	SetTextAttribute	35
4.3	Event	36
4.3.1	Clone	36
4.3.2	IsGeneric	36
4.3.3	Context	37
4.3.4	Description	37
4.3.5	IconID	37
4.3.6	Label	37
4.3.7	SubjectName	38
4.3.8	Time	38
4.4	EventContext	39
4.4.1	Colour	39
4.4.2	Description	39
4.4.3	IconID	40
4.4.4	Label	40
4.5	EventStore	41
4.5.1	AddEvent	41
4.5.2	Event	41
4.5.3	EventContext	42
4.5.4	EventContext Count	42
4.5.5	EventCount	42
4.5.6	RemoveEvent	42
4.5.7	RemoveEventAt	42
4.6	ForcePlate	43
4.6.1	AnalogToVideo Ratio	43
4.6.2	CenterOfPressure	43
4.6.3	Corners	44
4.6.4	FirstSampleNum	44
4.6.5	ForceUnits	44
4.6.6	GetCentreOf Pressures	44

4.6.7	GetForces	45
4.6.8	GetMoments	45
4.6.9	LastSampleNum	45
4.6.10	MomentUnits	45
4.6.11	PointUnits	45
4.6.12	Reaction	46
4.6.13	SampleRate	46
4.7	ParameterStore	47
4.7.1	Contains Parameter	48
4.7.2	ContainsSection	48
4.7.3	CreateBoolean Array	48
4.7.4	CreateColour Array	48
4.7.5	CreateDate TimeArray	49
4.7.6	CreateInteger Array	49
4.7.7	CreateRealArray	49
4.7.8	CreateTextArray	50
4.7.9	Delete Parameter	50
4.7.10	DeleteSection	50
4.7.11	GetBoolean	51
4.7.12	GetColour	51
4.7.13	GetDateTime	52
4.7.14	GetDimension Count	52
4.7.15	GetDimension	52
4.7.16	GetInteger	53
4.7.17	GetReal	53
4.7.18	GetText	53
4.7.19	SectionCount	53
4.7.20	SectionName	54
4.7.21	SetBoolean	54
4.7.22	SetColour	54
4.7.23	SetDateTime	55
4.7.24	SetDefault TextLength	55
4.7.25	SetInteger	55
4.7.26	SetReal	55
4.7.27	SetText	56
4.8	PECS	57
4.8.1	CapabilityFlags	57
4.8.2	ConnectionCount	57
4.8.3	Execute	58
4.8.4	Refresh	58
4.8.5	Supports CapabilityAll	58
4.8.6	Supports CapabilitySome	59
4.8.7	Processor	59
4.8.8	RefCounter	59
4.8.9	Trial	59

4.9	Processor	60
4.9.1	Log	60
4.9.2	SetStatus Message	60
4.10	Subject	61
4.10.1	DisplaySetName	61
4.10.2	LabelPrefix	61
4.10.3	MarkerSetName	62
4.10.4	ModelName	62
4.10.5	ModelParams Name	62
4.10.6	Name	62
4.10.7	SetLabel PrefixUsage	62
4.10.8	SetMarkerSet	63
4.11	Trajectory	64
4.11.1	ClearSelected Points	65
4.11.2	ClearTrajectory	65
4.11.3	FindGapAt	65
4.11.4	FindGap Backward	66
4.11.5	FindGap Forward	66
4.11.6	FirstValidField Num	67
4.11.7	GetPoint	67
4.11.8	GetPoints	67
4.11.9	GetPointCameras	67
4.11.10	HasEmptyLabel	68
4.11.11	Hide	68
4.11.12	InvalidatePoint	68
4.11.13	IsVisible	68
4.11.14	LastValidField Num	68
4.11.15	MatchesLabel	69
4.11.16	MatchesLabel Prefix	69
4.11.17	PointCamera	69
4.11.18	PointCamera Count	69
4.11.19	SampleRate	70
4.11.20	SelectedPoint Count	70
4.11.21	SelectedPoint Field	70
4.11.22	Units	70
4.11.23	SelectPoint	70
4.11.24	SetPoint	71
4.11.25	SetPoints	71
4.11.26	Show	71
4.11.27	Colour	71
4.11.28	Label	72
4.11.29	MarkerSize	72
4.11.30	TypeGroup	72

4.12 Trial 73
4.12.1 AnalogBaseRate 74
4.12.2 AnalogChannel 74
4.12.3 AnalogChannel Count 74
4.12.4 CreateTrajectory 75
4.12.5 DataPath 75
4.12.6 DateAndTime 75
4.12.7 DefaultMarker Size 75
4.12.8 DeleteAll Trajectories 75
4.12.9 DeleteAll Unlabelled Trajectories 76
4.12.10 DeleteTrajectory 76
4.12.11 Description 76
4.12.12 EclipseNode 76
4.12.13 EventStore 76
4.12.14 Execute 77
4.12.15 FindAnalog Channel 77
4.12.16 FindSubject 78
4.12.17 FindSubjectBy Prefix 78
4.12.18 FindTrajectory 78
4.12.19 FindTrajectory Index 79
4.12.20 FirstValid TrajectoryFieldNum 79
4.12.21 ForcePlate 79
4.12.22 ForcePlateCount 79
4.12.23 IsStatic 80
4.12.24 LastValid TrajectoryFieldNum 80
4.12.25 Notes 80
4.12.26 Parameter Store 80
4.12.27 ReferenceName 80
4.12.28 RefreshChanges 81
4.12.29 Register TrajectoryTypeGroup 81
4.12.30 SetDefault MarkerSize 81
4.12.31 SetLabelPrefix Usage 82
4.12.32 SetStaticStatus 82
4.12.33 SetTrajectory TypeGroupUnits 82
4.12.34 Subject 82
4.12.35 SubjectCount 83
4.12.36 Trajectory 83
4.12.37 TrajectoryCount 83
4.12.38 TrajectoryType GroupCount 83
4.12.39 TrajectoryType GroupName 83
4.12.40 TrajectoryType GroupUnits 84
4.12.41 Type 84
4.12.42 UsesLabel Prefixes 84
4.12.43 VideoFieldCount 84
4.12.44 VideoRate 84

5 ADDITIONAL UTILITY INTERFACES 85
5.1 Colour 85
5.1.1 Red 85
5.1.2 Green 85
5.1.3 Blue 85
5.2 Point 86
5.2.1 IsValid 86
5.2.2 X 86
5.2.3 Y 86
5.2.4 Z 86
5.3 Range 87
5.3.1 Begin 87
5.3.2 End 87
5.3.3 Size 87
5.4 Ratio 88
5.4.1 Denominator 88
5.4.2 Numerator 88
5.5 Reaction 89
5.5.1 ForceX 89
5.5.2 ForceY 89
5.5.3 ForceZ 89
5.5.4 IsValid 89
5.5.5 MomentX 90
5.5.6 MomentY 90
5.5.7 MomentZ 90
5.6 Reconstruction 91
5.6.1 Camera 91
5.6.2 CameraCount 91
5.6.3 IsValid 92
5.6.4 Residual 92
5.6.5 X 92
5.6.6 Y 92
5.6.7 Z 92
5.7 RefCounter 93
5.7.1 Count 93
5.7.2 Decrement 93
5.7.3 Increment 93

1 INTRODUCTION AND REQUIREMENTS

The Pipeline External Communication Server, or **PECS**, is an extension to Workstation that provides an ActiveX interface for other programs. The application is an "automation server", which means that it can be automatically run and operated from the system registry. Furthermore, the PECS is a "single use" server, meaning that many different processes can connect to the same server, and hence the same Workstation.

To use PECS, you will require Workstation version 4.5, the Workstation SDK version 2.0 and a valid hardware lock. The server is provided with the PECS Workstation Plug-In for operation.

2 INSTALLING THE PECS SERVER

2.1 Installation and Registration

The PECS installer copies the following files on to your computer in your Vicon installation directory:

...\Vicon\Workstation\PECS.exe
...\Vicon\Workstation\PECS.tlb
...\Vicon\Workstation\PECS.txt
...\Vicon\PlugIns\PECSPlugIn.ini
...\Vicon\PlugIns\PECSPlugIn.txt
...\Vicon\PlugIns\PECSPlugIn.vpi
...\Vicon\PlugIns\PECS Plug-In Options\Example.car2
...\Vicon\PlugIns\Example.xls
...\Vicon\PlugIns\Example.doc
...\Vicon\PlugIns\Example.m
...\Vicon\PlugIns\Example.vi
...\Vicon\Manuals\PECS_v1_0.pdf (this document)

Shortcuts to this manual and an uninstall program will also be placed in your start menu.

The installer also performs all of the necessary registration of the PECS server with your computer's registry. If you need to remove PECS from your computer, run the Uninstall PECS program from your Start Menu\Programs\Vicon\PECS folder. Alternatively, PECS can be removed using the Windows control panel Add/Remove Programs utility.

2.2 Installing the Server in Client Applications

In addition to system installation, several client applications and development environments will require specific steps to be taken to give the application access to PECS.

2.2.1 Microsoft Office Products

Microsoft Office applications have access to ActiveX objects via VBA (Visual Basic for Applications). Section 2.2.2 covers installation of the PECS server in Visual Basic and VBA.

Support is provided for running VBA scripts within Word© and Excel© and your security settings may need to be adjusted.

Office 98/2000

1. Open the Office application in question. On the **Tools** menu, click **Macro**, and then click **Security** to open the **Macro Security** dialog box.
2. On the **Security Level** tab, make sure that macro security is either **Medium** or **Low**.
3. Click **OK** to apply the setting and close the Office Application.

Office XP

If the default security settings are not correct, the following error message will be seen when attempting to run a VBA script in a Word or Excel document:

"Programmatic Access to Visual Basic Project is not trusted"

To change the security setting, follow these steps:

1. Open the Office application in question. On the **Tools** menu, click **Macro**, and then click **Security** to open the **Macro Security** dialog box.
2. On the **Security Level** tab, make sure that macro security is either **Medium** or **Low**.
3. On the **Trusted Sources** tab, click to select the **Trust access to Visual Basic Project** check box to turn on access.
4. Click **OK** to apply the setting and close the Office Application.

2.2.2 Visual Basic

The PECS server must be selected as a reference in the Visual Basic development environment before it can be accessed within Visual Basic (or VBA) code. To do this:

1. Open the Visual Basic (or VBA) Project needing PECS.
2. On the **Project** menu (**Tools** menu in VBA), click **References** to open the references dialog box.
3. If the server has been registered, PECS should appear in the list.
4. If the server has not been registered, click **Browse** and search for PECS.TLB (which should be in your Vicon\Workstation folder).
5. Check the PECS entry in the reference list.
6. Click **OK** to apply the settings.

2.2.3 Microsoft Visual C++

Microsoft Visual C++ does not require any explicit installation of the PECS server: Once it has been registered on your system, it will be available using the appropriate C++ calls.

However, C++ wrapper classes can be generated by Visual Studio using the Class Wizard:

1. Open the class wizard by right clicking in the workspace and selecting **Class Wizard**
2. Click **Add Class** and Select **From Type Library**
3. Browse for PECS.TLB (which should be in your Vicon\Workstation folder).
4. Select the interface classes you wish to use from the class list, enter the name of the .cpp and .h file you wish to use and click **OK**.

Note: The list of interface classes shown on the type library import dialog reflects both parts of the dual interface provided by PECS. Depending on user requirements, it is likely that only one of the interface definitions will be needed. For example, `IPECSDoc` and `IDualPECSDoc` will be listed, but it will most likely only be necessary to select `IPECSDoc` for wrapping. This interface structure is introduced in Section 3.3.

2.2.4 Microsoft Visual J++

The PECS server can be incorporated into your Microsoft Visual J++ project using automatically created class wrappers that provide the interface for accessing the ActiveX interfaces. These class wrappers will be added to packages in your project directory.

To insert the PECS object in the development environment:

1. On the **Project** menu, click **Add COM Wrapper** to display the **COM Wrappers** dialog box.
2. If the server has been registered, PECS should appear in the list of COM objects.
3. If the server has not been registered, click **Browse** and search for PECS.TLB (which should be in your Vicon\Workstation folder).
4. Check the PECS entry in the COM objects list.
5. Click **OK**.

Note: You can avoid wrapping the object for each project you create by creating the wrapper classes in a single project and placing referencing these classes using the project **Classpath**.

2.2.5 MATLAB

MATLAB prior to release 12.0 and release 14 onwards

Once the PECS server has been registered on your PC, MATLAB will be able to access the server and its interfaces using ActiveX.

MATLAB release 12.1 to release 13

Because of a change made to MATLAB in the way the ActiveX objects are created in release 12.1, the PECS server cannot be accessed using the same ActiveX methods.

A fix is provided for this problem with the PECS installation and is contained within MATLABFix.exe. This self extracting executable will update the following files in your existing MATLAB installation, allowing it to access ActiveX objects in the same way as release 12.0 (and before):

...\MATLAB\bin\win32\comcli.dll
...\MATLAB\toolbox\matlab\winfun\comcli\private\newprogid.m

More information on this change to MATLAB release 12.1 is available at:

http://www.mathworks.com/support/solutions/data/32562.shtml

2.2.6 LabVIEW

When designing a Virtual Instrument that requires access to the PECS server, an automation object of the PECS.Document type needs to be created. To do this:

1. Insert an Automation Refnum from the **All Controls\Refnum controls** palette.
2. Right click on the control and click **Select ActiveX Class\Browse** to open the ActiveX control selection dialog box.
3. Select PECS from the drop down list of ActiveX objects and click **Show Creatable Objects Only**.
4. Expand the PECS (PECS.Document) in the objects window, select IDualPECSDoc and click **OK**.

All interface methods and properties should now be easily accessible using the LabVIEW user interface.

3 USING THE AUTOMATION SERVER

The figure below illustrates the operation of the PECS in conjunction with Workstation, the Plug-In and an example client process.

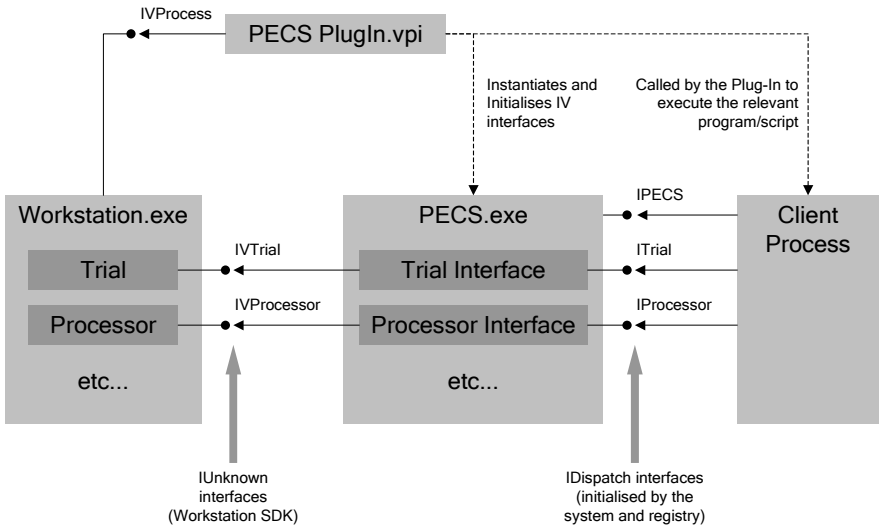

Figure 1. Server Operation.

First of all, when the Plug-In is executed (in the Workstation pipeline), it requests an instance of PECS from the registry and connects the Workstation SDK interface to it. Finally, the Plug-In starts the client process and instructs it to evaluate the required program/script (specified in the Plug-In options). The client process then takes over, with the various calls to the PECS in the client program/script being routed through the server to Workstation.

3.1 Plug-In Options

The options for the PECS Plug-In are shown below in Figure 2.

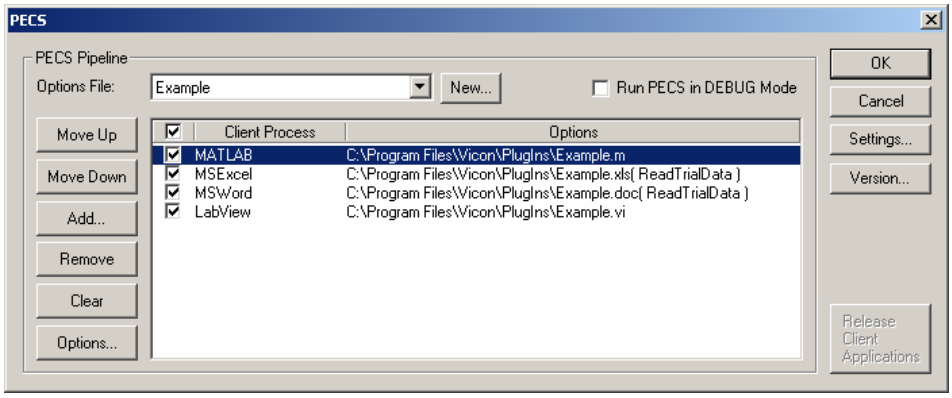

Figure 2. Main Plug-In Options.

The Plug-In interface allows the creation of a number of pipelines that incorporate ActiveX clients. In addition, the settings for client applications can be changed.

3.1.1 Using Options Files

The options for each defined pipeline are contained within options files located in the folder:

...\Vicon\PlugIns\PECS Plug-In Option

These files have the extension car2 ***and should not be manually edited***. The Plug-In installation is provided with an example setup called **Default** which should be loaded by default whenever the Plug-In used for the first time on a trial. To create a new pipeline options file, click **New...** on the options dialog, which will open the dialog box shown below in Figure 3.

Figure 3. Creating a New Options File.

Type the name of the pipeline you wish to create in this box and click **OK**. Click **Cancel** if you don't wish to create a new options file.
To select from the available pipeline options files available, click the drop **Options File** drop down list.

3.1.2 Managing the Pipeline

The PECS pipeline window indicates the processes defined in the pipeline, whether they can be run and whether they are enabled. Processes that are unable to run for some reason (for example not having MATLAB installed on your machine) will appear in red. Checking the box to the left of a process enables this process for running in the same way as the normal Workstation pipeline.

Processes can be added to and removed from the pipeline using the **Add...** and **Remove** buttons and their order can be changed using the **Move Up** and **Move Down** buttons. To empty a pipeline of processes, click the **Clear** button.

Clicking the **Options...** button or double clicking an entry in the pipeline will open that client processes options dialog.

3.1.3 Adding Processes to the Pipeline

When **Add...** is clicked on the main Plug-In options dialog, the dialog shown below in Figure 4 will be shown.

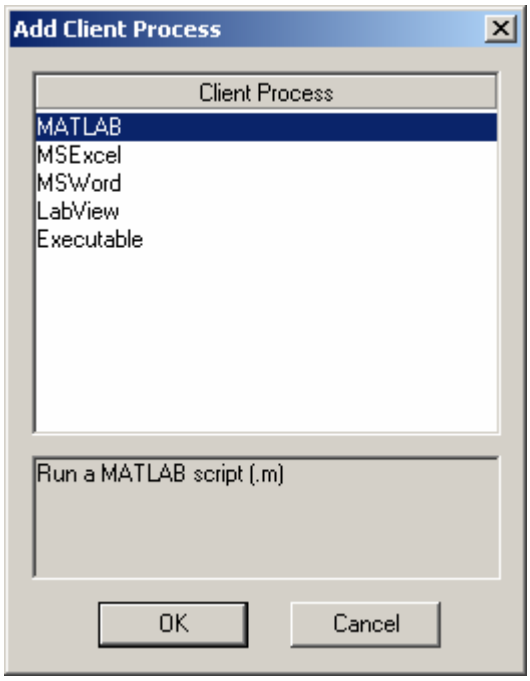

Figure 4. Adding a Process to the Pipeline.

The list of client processes available for evaluation in the pipeline on your computer will depend on what client applications you have installed. For instance if you have Microsoft Office, but neither MATLAB nor LabVIEW, this list will only contain **MSExcel**, **MSWord**, **Executable** and **Debugging**.

To add a process, select the appropriate process from the list and click **OK** or double click the process. This will then open the process options

dialog. Click **Cancel** if you don't wish to add a new process to your pipeline.

3.1.4 Process Options

Each client process has a specific set of options and these are summarised in the following sections.

Microsoft Office Products

Figure 5 and Figure 6 below show the options dialog boxes for adding these two Office products.

Figure 5. Adding a Microsoft Excel Process to the Pipeline.

Figure 6. Adding a Microsoft Word Process to the Pipeline.

In both cases, the document to be used must be located by clicking browse(...) button next to or typing the full path of the document in the top edit box.

Once the document has been located, clicking the browse (...) button next to the **Sub-Routine** edit box will show a list of the VBA sub-routines available in that document as shown below in Figure 7.

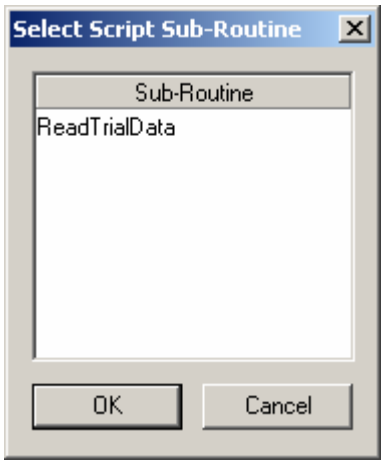

Figure 7. Select a Sub-Routine Defined in a Document.

Either select a sub-routine from this list and click **OK** or double click the required sub-routine. Alternatively, type the name of the sub-routine you wish to run in the **Sub-Routine**.

The **Launch in Single Use Mode** check box for Microsoft Office client processes allows batch processing to run a macro in the same document, whilst leaving it open. When batch processing is complete, the Office application and document will remain open. Operating in single use mode requires the user to Release Client Applications once batch processing is complete

MATLAB The options for a MATLAB process are shown below in Figure 8.

Figure 8. Adding a MATLAB Process to the Pipeline.

To select a MATLAB script (.m file) for processing, either type the full path and name in the **MATLAB Script** edit box or click browse (**...**) and locate it.

The **Launch MATLAB in Single Use Mode** check box allows evaluation of the MATLAB script(s) in an open automation MATLAB application. When processing is complete, MATLAB will remain open and will require the user to Release Client Applications once all operations in MATLAB have been completed.

LabVIEW The options for a LabVIEW Virtual Instrument are shown below in Figure 9.

Figure 9. Adding a LabVIEW Process to the Pipeline.

To select a Virtual Instrument (.vi file) for processing, either type the full path and name in the **Virtual Instrument** edit box or click browse (**...**) and locate it.

The **Launch LabVIEW in Single Use Mode** allows batch processing to run a Virtual Instrument in LabVIEW, whilst leaving it open. When batch processing is complete, LabVIEW will remain open. Operating in single use mode requires the user to Release Client Applications once batch processing is complete

Standalone Executables

The options for a standalone executable client process are shown below in Figure 10.

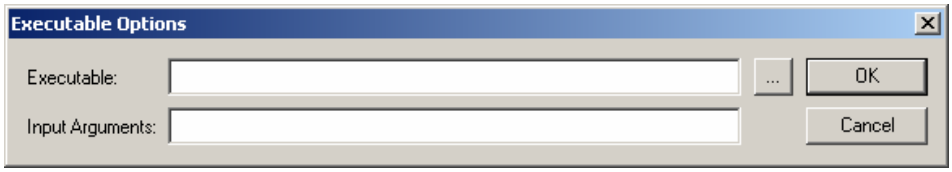

Figure 10. Adding a Standalone Executable Process to the Pipeline.

Running standalone executables is slightly different than for all other client applications in that they are not activated using ActiveX.

Select the executable you wish to run by typing its name in the **Executable** edit box or click browse (**...**) and locate it. In addition, any required command lines arguments required should be typed in the **Input Arguments** edit box, as they would be typed at the DOS command line.

For example, it would be possible to run the example MATLAB script using this option by selecting the executable:

\Matlab\bin\win32\MATLAB.exe

and entering the input arguments:

/r <Vicon Path>\PlugIns\Example.m

3.1.5 Debugging Client Code

The **Run PECS in DEBUG Mode** switch on the main Plug-In options allows client applications to be run in their native debug mode.

In this mode of operation, the Plug-In launches the PECS server and waits for the client process to connect to PECS and then stops processing when the client process releases PECS.

To debug the example Excel VBA macro for example:

1. Open a trial or select a single trial in the Eclipse window.
2. Open the PECS Plug-In options from the pipeline.
3. Click **Run PECS in DEBUG Mode** and then **OK**.
4. Click **Process Now** on the pipeline pane. The PECS server will then be launched and awaits connection from the client application.
5. Open Example.xls. On the **Tools** menu, click **Macro**, and then click **Visual Basic Editor** to open the VBA project
6. Set a breakpoint near the beginning of the program then select **Run Sub/UserForm** from the **Run** menu.
7. Step through the code as desired.

3.1.6 Settings

Clicking **Settings...** on the main Plug-In options dialog will open the dialog shown in Figure 11 below.

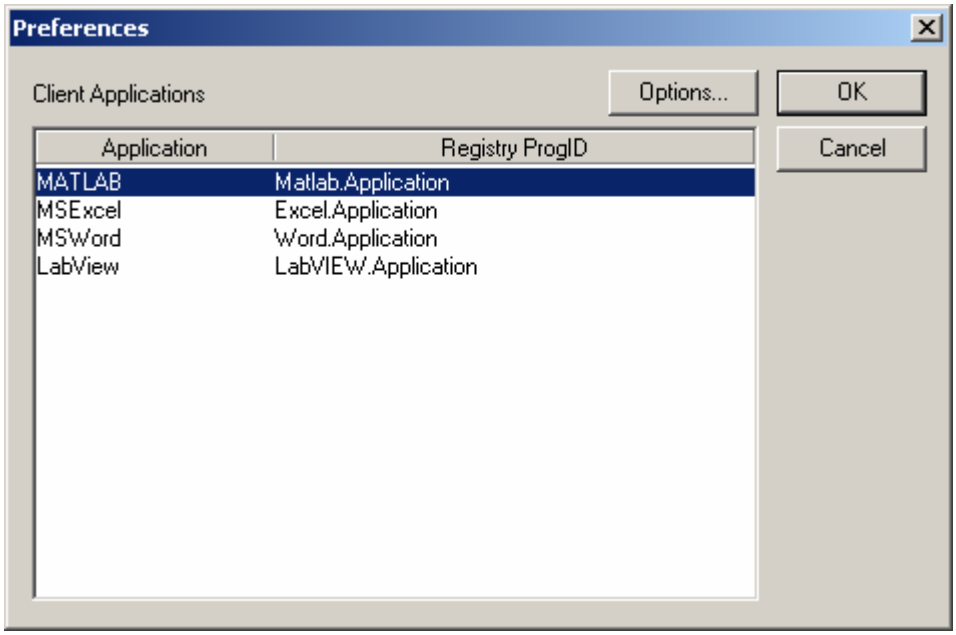

Figure 11. Settings.

The list of **Client Applications** in Figure 11 indicates the unique **Registry ProgID** (program identifier) that will be used to activate the various client processes by ActiveX and it is unlikely that these will need changing.

By selecting a client application in the list and clicking **Options...** or double clicking the application, the registry ProgID can be edited in the dialog shown in Figure 12 below.

Figure 12. Changing an ActiveX Program ID.

3.1.7 Product Version Information

To check the version of the PECS Plug-In and sever you have installed, click **Version...** on the main Plug-In Options dialog. This will open the information dialog shown in Figure 13.

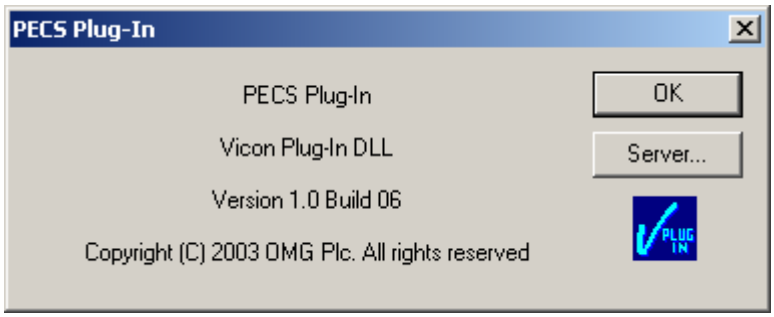

Figure 13. Plug-In Version Dialog.

Clicking the **Server...** button will show the PECS server information dialog shown in Figure 15.

3.1.8 Releasing Client Applications

When processes are run in **Single Use** mode, after processing, the **Release Client Applications** button will become enabled. This is because the Plug-In still "owns" an ActiveX interface for the client process(s).

Clicking the **Release Client Applications** button causes these interfaces to be released. In the case of MATLAB, this will close the open MATLAB workspace, but all other applications will remain open.

3.2 Server Options

When the PECS server is in operation, it will be displayed on top of all other open windows on your desktop as shown in Figure 14.

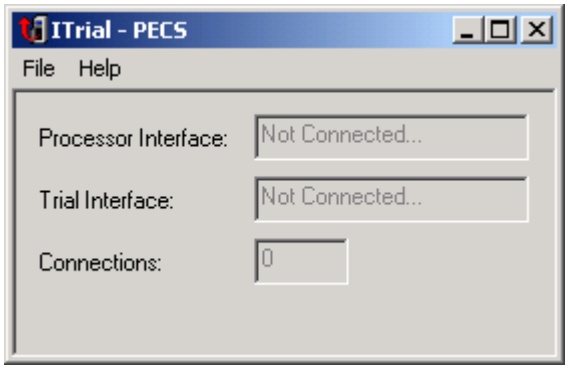

Figure 14. Server Dialog

The majority of information displayed on the PECS server is for information only.

Processor Interface:
This indicates whether there is a connection between the PECS server and the Workstation processor. The Processor interface is the interface that writes messages in the processing log and can update the Workstation status bar message.

Trial Interface:
This indicates whether there is a connection between the PECS server and the Workstation trial. The Trial interface is the main interface through which the majority of the PECS interface functionality can be called. (See the description of the Interface Hierarchy)

Connections:
This indicates how many external processes are connected to the PECS interface. During processing and debugging this will invariably be 1.

The PECS server can be closed down by selecting **Exit** from the **File** menu or using the top right icon, although this is not recommended if a client process may still be accessing the server interface.

Selecting **About PECS** from the **Help** menu will show the version information for the server shown below in Figure 15.

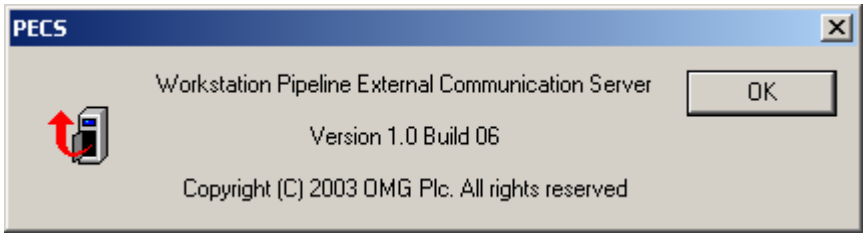

Figure 15. Server Version Dialog.

3.3 Writing Client Processes using the ActiveX Interface

When a client application creates an instance of an ActiveX PECS interface, all of the interface classes required to operate Workstation can be obtained through the hierarchy shown below in Figure 16.

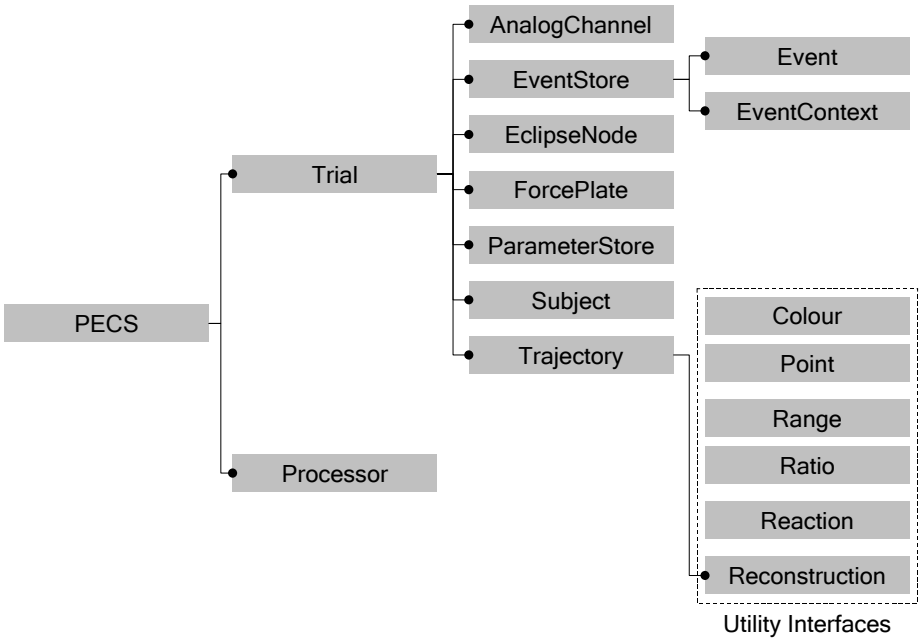

Figure 16. Interface Hierarchy.

The pattern for any client process, whether a script, Virtual instrument or a standalone executable written in some other programming language is as follows:

1. Obtain a PECS interface.
2. Perform necessary processing (using other interfaces as required).
3. After processing has been finished, **Release** the PECS interface.

When interfaces other than PECS are used during processing, they must each be **Released** when they are no longer required.

Releasing the PECS interface last is vital, as any subsequently opened PECS interfaces will NOT be attached to Workstation.

The syntax for obtaining the PECS interface is different for each client application and is summarised in the Sample code section.

3.3.1 The 'DUAL' ActiveX Interface

Each of the interfaces in Figure 16 are all provided as "Dual" interfaces. This ensures that all client applications are capable of obtaining and operating these interfaces. Because of this, in many applications such as Visual Basic and LabVIEW, there will be two interface versions available for each. For example:

PECS:

- IPECSDoc
- IDualPECSDoc

Trial:

- ITrial
- IDualTrial

It is recommended that the "Dual" implementation of these interfaces be used at all times.

3.3.2 Features not for External Use

The following interface methods are not intended for use in client code:

Set PECS::Processor
Set PECS::Trial
Set PECS::RefCounter
Method RefCounter::Increment
Method RefCounter::Decrement

3.4 Sample Code

The following code snippets and instructions summarise how to obtain the PECS server, call methods, get and set properties on a generic interface. In addition, arguments to be passed as arrays are introduced.

3.4.1 Visual Basic (and VBA)

Create Server

```
Dim Server As Object
Set Server = CreateObject("PECS.Document")
```

Call Method (with return)

```
ReturnValue = Interface.Method( InputArg,
... )
```

Call Method (as interface)

```
Dim Interface As Object
Set Interface = Interface.Method( InputArg,
... )
```

Call Method (no return)

```
Call Interface.Method( InputArg, ... )
```

Get Property

```
ReturnValue = Interface.Property()
```

Get Property (as interface)

```
Dim Interface As Object
Set Interface = Interface.Method()
```

Set Property

```
Interface.Property() = Value
```

Set Property (as interface)

```
Dim Interface As Object
Set Interface.Property() = Interface
```

Release

```
Interface.Release()
```

Array Argument

```
Dim Argument As Variant
Argument = Array( 1, 2, 3 )
```

3.4.2 MATLAB

Create Server

```
hServer = actxserver( 'PECS.Document" );
```

Call Method

```
ReturnValue = invoke( Interface, 'Method',
'InputArg', ... );
```

Get Property

```
ReturnValue = get( Interface, 'Property' );
```

Set Property

```
set( Interface, 'Property', Value );
```

Release

```
release( Interface );
```

Array Argument

```
Argument = [ 1, 2, 3 ]
& OR Argument = { 1, 2, 3 }
```

3.4.3 LabView

Create Server

1. Insert an **Automation Refnum** object.
2. Right click the control, click **Select ActiveX class**, and select or browse for PECS::IDualPECSDoc.
3. Insert an **Automation Open** block and connect the PECS automation refnum to its input.

Call Method

1. Insert an **Invoke Node**.
2. Connect the PECS automation refnum to its input.
3. Select the appropriate method from the drop down list.

Get Property

1. Insert a **Property Node**.
2. Connect the PECS automation refnum to its input.
3. Select the appropriate property from the drop down list.

Set Property

1. Insert a **Property Node**.
2. Connect the PECS automation refnum to its input.
3. Select the appropriate property from the drop down list.
4. Right click on the property block and select **Change All To Write**.
5. Connect the required value/automation refnum to the property box.

Release

1. Insert a **Close Reference** block
2. Connect the PECS automation refnum to its input.

Array Argument

1. Insert an **Array Constant** block of the appropriate dimension.
2. Add a number of **Constant Elements** and set them to the required value.
3. Insert a **To Variant** block.
4. Connect the **Array Constant** output to the **To Variant** input.
5. Use the **To Variant** output as the array argument.

Further example code is provided with the installation in the files:

...\Vicon\PlugIns\Example.xls	(VBA)
...\Vicon\PlugIns\Example.doc	(VBA)
...\Vicon\PlugIns\Example.m	(MATLAB)
...\Vicon\PlugIns\Example.vi	(LabVIEW)

Please refer to your client application documentation for further assistance.

4 ALPHABETIC INTERFACE REFERENCE

All of the interface declarations in this section are for the Dual Interfaces and the Interface Description Language is used. Although the return value of all of the functions is technically a standard COM return value (HRESULT), the return values in the method and property descriptions are those specified as [out, retval] (output, returned value). The exact syntax for calling methods and properties will differ between client applications and is covered in the Sample Code section.

4.1 AnalogChannel

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents the stream of data from a single analog channel. This interface is rarely implemented by plug-ins but, instead, plug-ins typically obtain a pointer to an analog channel object through other means such as from a trial object (see ITrial interface).
Methods:	1. AnalogToVideoRatio 2. BaseRateDivider 3. FirstSampleNum 4. GetSample 5. GetSamples 6. LastSampleNum 7. SampleRate 8. SetSample 9. SetSamples 10. Units 11. MatchesLabel
Properties:	12. Label

4.1.1 AnalogToVideoRatio

Declaration: HRESULT AnalogToVideoRatio([out, retval]Ratio** retval)

Return type: Ratio — Ratio of analog to video rates

Parameters: NONE

Description: Obtains the ratio of the analog to video capture rates for this channel. Call Release on the returned object context when it is no longer required.

4.1.2 BaseRateDivider

Declaration: HRESULT BaseRateDivider([out, retval]long* retval)

Return type: long — Base rate divider

Parameters: NONE

Description: Obtains the analog base sample frequency divider for this channel. All analog channels in Vicon are captured at the same base frequency but, if the functionality is available, each channel may reduce its sampling rate by a fixed integer ratio represented by this divider. Typically, the value will be one for all channels.

4.1.3 FirstSampleNum

Declaration: HRESULT FirstSampleNum([out, retval]long* retval)

Return type: long — last valid sample number

Parameters: NONE

Description: Returns the first sample index for which data is available on this channel.

4.1.4 GetSample

Declaration: HRESULT GetSample([in]long i_Index, [out, retval]double* retval)

Return type: double — Value at this sample

Parameters: long i_Index — Sample index of data to get

Description: Gets the analog channel data value at the specified index. Because of storage constraints, the data may be quantised before storage.

4.1.5 GetSamples

Declaration: HRESULT GetSamples([in]long i_Begin, [in]long i_End, [out, retval]VARIANT* retval)

Return type: VARIANT — N vector of sample values

Parameters: long i_Begin — Range begin sample
long i_End — Range end sample

Description: Obtains the values of the analog channel over the specified one-based sample range (inclusive of end sample) as an N vector.

4.1.6 LastSampleNum

Declaration: HRESULT LastSampleNum([out, retval]long* retval)

Return type: long — Last valid sample number

Parameters: NONE

Description: Returns the last, inclusive sample index for which data is available on this channel.

4.1.7 SampleRate

Declaration: HRESULT SampleRate([out, retval]double* retval)

Return type: double — Sample rate

Parameters: NONE

Description: Returns the sample rate of the data for this analog channel, in Hz.

4.1.8 SetSample

Declaration: HRESULT SetSample([in]long i_Index, [in]double i_Value)

Return type: NONE

Parameters:

long i_Index	Sample index of data to set
double i_Value	New value for this sample

Description: Sets the analog channel data value at the specified index. Because of storage constraints, the data may be quantised before storage.

4.1.9 SetSamples

Declaration: HRESULT SetSamples([in]long i_Begin, [in]long i_End, [in]VARIANT* i_Points)

Return type: NONE

Parameters:

long i_Begin	Range begin field number
long i_End	Range end field number
VARIANT* i_Points	3 x N array of co-ordinates

Description: Sets the values of the analog channel over the specified one-based sample range (inclusive of end sample) as an N vector of values. If the vector passed in does not have the same number of elements as there are samples in the specified range, the server throws an exception.

4.1.10 Units

Declaration: HRESULT Units([out, retval]BSTR* retval)

Return type: BSTR — String containing units

Parameters: NONE

Description: Returns the analog channel units in the form of a text string.

4.1.11 MatchesLabel

Declaration: HRESULT MatchesLabel([in]BSTR i_Label, [out,

retval]BOOL* retval)

Return type:	BOOL	Boolean for match
Parameters:	BSTR i_Label	Label to match

Description: Convenient way of directly checking for a match against a given label rather than using GetLabel and checking that manually.

Returns TRUE if the label matches, otherwise FALSE.

4.1.12 Label

Declaration: [propput] HRESULT Label([in]BSTR i_Label)
[propget] HRESULT Label([out, retval]BSTR* retval)

Return type:	BSTR	Channel label
Parameters:	BSTR i_pLabel	New channel label

Description: Sets and gets the analog channel label in the form of a text string.

4.2 EclipseNode

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	An Eclipse Node is normally a component associated with another object such as a Trial. The interface provides access to the node attributes.
Methods:	1. DataPath 2. NodeID 3. Title 4. GetBooleanAttribute 5. GetDateAndTimeAttribute 6. GetIntegerAttribute 7. GetRealAttribute 8. GetTextAttribute 9. SetBooleanAttribute 10. SetDateAndTimeAttribute 11. SetIntegerAttribute 12. SetRealAttribute 13. SetTextAttribute
Properties:	

4.2.1 DataPath

Declaration: HRESULT DataPath([out, retval]BSTR* retval)

Return type: BSTR Data path

Parameters: NONE

Description: All Eclipse nodes have an associated file system data path. Use this method to retrieve it in the form of a text string.

4.2.2 NodeID

Declaration: HRESULT NodeID([out, retval]long* retval)

Return type: long Node ID

Parameters: NONE

Description: Obtains the unique ID associated with the node. Note that this ID is only valid during execution of the application and should not be persisted anywhere.

4.2.3 Title

Declaration: HRESULT Title([out, retval]BSTR* retval)

Return type: BSTR — Node title

Parameters: NONE

Description: All Eclipse nodes have an associated title. This is typically the base name for associated files. Use this method to retrieve the title of the node in the form of a text string.

4.2.4 GetBoolean Attribute

Declaration: HRESULT BooleanAttribute([in]BSTR i_Name, [out, retval]BOOL* retval)

Return type: BOOL — Attribute boolean value

Parameters: BSTR i_Name — Attribute name

Description: Obtains the value of the specified attribute as a boolean. If the attribute has not been set or is not a Boolean then the server will throw an exception.

4.2.5 GetDateAnd TimeAttribute

Declaration: HRESULT DateAndTimeAttribute([in]BSTR i_Name, [out, retval]DATE* retval)

Return type: DATE — Attribute date structure

Parameters: BSTR i_Name — Attribute name

Description: Obtains the value of the specified attribute as a set of date and time values. If the attribute has not been set or does not contain date and time information then the server will throw an exception.

The date is a standard COM DATE type. This is a floating point number which represents days as whole number increments starting from midnight 30th December 1899.

4.2.6 GetInteger Attribute

Declaration: HRESULT IntegerAttribute([in]BSTR i_Name, [out, retval]long* retval)

Return type: long — Attribute integer value

Parameters: BSTR i_Name — Attribute name

Description: Obtains the value of the specified attribute as a long integer. If the attribute has not been set or cannot be converted to an integer then the server will throw an exception.

4.2.7 GetRealAttribute

Declaration: HRESULT RealAttribute([in]BSTR i_Name, [out, retval]double* retval)

Return type: double — Attribute double value

Parameters: BSTR i_Name — Attribute name

Description: Obtains the value of the specified attribute as a double precision real. If the attribute has not been set or cannot be converted to a real then the server will throw an exception.

4.2.8 GetTextAttribute

Declaration: HRESULT TextAttribute([in]BSTR i_Name, [out, retval]BSTR* retval)

Return type: BSTR — Attribute text value

Parameters: BSTR i_Name — Attribute name

Description: Obtains the value of the specified attribute as text into a text buffer object. If the attribute has not been set then the server will throw an exception.

4.2.9 SetBoolean Attribute

Declaration: HRESULT BooleanAttribute([in]BSTR i_Name, [in]BOOL i_Value)

Return type: NONE

Parameters: BSTR i_Name — Attribute name
BOOL i_Value — New attribute boolean value

Description: Sets the value of the specified attribute as a boolean.

4.2.10 SetDateAnd TimeAttribute

Declaration: HRESULT DateAndTimeAttribute([in]BSTR i_Name, [in]DATE i_Date)

Return type: NONE

Parameters: BSTR i_Name — Attribute name
DATE i_Date — New attribute date structure

Description: Obtains the value of the specified attribute as a set of date and time values.

The date is a standard COM DATE type. This is a floating point number which represents days as whole number increments starting from midnight 30th December 1899. If an invalid DATE is passed into this function, the server will throw an exception.

4.2.11 SetInteger Attribute

Declaration: HRESULT IntegerAttribute([in]BSTR i_Name, [in]long i_Value)

Return type: NONE

Parameters:

BSTR i_Name	Attribute name
long i_Value	New attribute integer

Description: Obtains the value of the specified attribute as a long integer.

4.2.12 SetRealAttribute

Declaration: HRESULT RealAttribute([in]BSTR i_Name, [in]double i_Value)

Return type: NONE

Parameters:

BSTR i_Name	Attribute name
double i_Value	New attribute double value

Description: Obtains the value of the specified attribute as a double precision real.

4.2.13 SetTextAttribute

Declaration: HRESULT TextAttribute([in]BSTR i_Name, [in]BSTR i_Value)

Return type: NONE

Parameters:

BSTR i_Name	Attribute name
BSTR i_Value	New attribute text value

Description: Obtains the value of the specified attribute as text into a text buffer object.

4.3 Event

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Event objects represent single time events within a trial. An example of an event is a foot strike against the floor during a walking trial. Every event must belong to an associated context, which might relate to a particular side (left or right) of the body for example.
Methods:	1. Clone 2. IsGeneric
Properties:	3. Context 4. Description 5. Icon 6. IconID 7. Label 8. SubjectName 9. Time

4.3.1 Clone

Declaration: HRESULT Clone([in]EventContext* i_pContext, [out, retval]Event ** retval)

Return type:	Event	Created event
Parameters:	EventContext i_Context	Pointer to context to create event in

Description: Creates a new event as an exact copy of the event to which this is applied, but places it in the given context which may differ from the original event. Call Release on the returned object when it is no longer required.

This method is mostly used by applications with an interactive user interface for manually creating events from a palette or from existing events. It may also be used by autocorrelation software for creating new events based on existing events.

4.3.2 IsGeneric

Declaration: HRESULT IsGeneric([out, retval]BOOL* retval)

Return type:	BOOL	Flag for generic event
Parameters:	NONE	

Description: Returns TRUE if the event is a generic one or FALSE if it corresponds to a specific and well defined event. Generic events are like user-defined events with editable labels and descriptions. Non-generic events

tend to be defined by the system, procedure or application and have fixed labels and descriptions that should not be modified.

4.3.3 Context

Declaration: [propput] HRESULT Context([in]EventContext * i_EventContext)
[propget] HRESULT Context([out, retval]EventContext ** retval)

Return type:	EventContext	Event context
Parameters:	EventContext i_EventContext	New event context

Description: Sets and gets the event context to which this event belongs. Call Release on the returned object when it is no longer required.

4.3.4 Description

Declaration: [propput] HRESULT Description([in]BSTR i_Description)
[propget] HRESULT Description([out, retval]BSTR* retval)

Return type:	BSTR	Event description
Parameters:	BSTR i_Description	New event description

Description: Sets and gets the event description in the form of a text string.

4.3.5 IconID

Declaration: [propput] HRESULT IconID([in]long i_IconID)
[propget] HRESULT IconID([out, retval]long* retval)

Return type:	long	Event icon identifier
Parameters:	long i_IconID	New event icon identifier

Description: Sets and gets an identifier for the event icon. This Icon ID is also used to represent the type of the event in a language independent way. Event types, and therefore icon IDs, should have carefully chosen values to ensure that they are unique among applications. They can be used to supply an alternative representation than the icon provided by Icon().

4.3.6 Label

Declaration: [propput] HRESULT Label([in]BSTR i_Label)
[propget] HRESULT Label([out, retval]BSTR* retval)

Return type:	BSTR	Event label
Parameters:	BSTR i_Label	New event label

Description: Sets and gets the event label in the form of a text string.

4.3.7 SubjectName

Declaration: [propput] HRESULT SubjectName([in]BSTR i_Name)
[propget] HRESULT SubjectName([out, retval]BSTR* retval)

Return type:	BSTR	Subject name
Parameters:	BSTR i_Name	New subject name

Description: Sets and gets the subject name in the form of a text string.

In multiple subject trials, events may be assigned to specific subjects. In single subject trials or if the event applies globally, then the subject name may be unspecified (blank).

4.3.8 Time

Declaration: [propput] HRESULT Time([in]double i_Time)
[propget] HRESULT Time([out, retval]double* retval)

Return type:	double	Event time (in seconds)
Parameters:	double i_Time	New event time (in seconds)

Description: Sets and gets the time of the event. This is specified in seconds from the start of the trial capture. Field or frame 1 corresponds to time 0.0s. To convert between time and frame representations, the data frame rate is required.

4.4 EventContext

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Each event must belong to an event context. The context allows certain attributes of groups of events to be shared. This can be important for maintaining consistency and association of objects in a graphical user interface. Example contexts include different sides of the body (i.e. left and right or front and rear).
Methods:	
Properties:	1. Colour 2. Description 3. Icon 4. IconID 5. Label

4.4.1 Colour

Declaration: [propput] HRESULT Colour([in]Colour * i_Colour)
[propget] HRESULT Colour([out, retval]Colour ** retval)

Return type:	Colour	Event context colour
Parameters:	Colour i_Colour	New event context colour

Description: Sets and gets the colour associated with the event context. Call Release on the returned object when it is no longer required.

4.4.2 Description

Declaration: [propput] HRESULT Description([in]BSTR i_Description)
[propget] HRESULT Description([out, retval]BSTR* retval)

Return type:	BSTR	Event context description
Parameters:	BSTR i_Description	New event context description

Description: Sets and gets the event context description in the form of a text string.

4.4.3 IconID

Declaration: [propput] HRESULT IconID([in]long i_IconID)
[propget] HRESULT IconID([out, retval]long* retval)

Return type:	long	Event icon identifier
Parameters:	long i_IconID	New event icon identifier

Description: Sets and gets an identifier for the event context icon. This Icon ID is also used to represent the type of the event context in a language independent way. Event context types, and therefore icon IDs, should have carefully chosen values to ensure that they are unique among applications. They can be used to supply an alternative representation than the icon provided by GetIcon.

4.4.4 Label

Declaration: [propput] HRESULT Label([in]BSTR i_Label)
[propget] HRESULT Label([out, retval]BSTR* retval)

Return type:	BSTR	Event label
Parameters:	BSTR i_Label	New event label

Description: Sets and gets the event context label in the form of a text string.

4.5 EventStore

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Typically owned by a trial, an event store object holds all event contexts and events associated with that trial.
Methods:	1. AddEvent 2. Event 3. EventContext 4. EventContextCount 5. EventCount 6. RemoveEvent 7. RemoveEventAt
Properties:	

4.5.1 AddEvent

Declaration: HRESULT AddEvent([in]BSTR i_Label, [in]BSTR i_ContextLabel, [out, retval]Event ** retval)

Return type: Event — Newly created event

Parameters: NONE

Description: Creates a new event with the given label and in the context identified by its given label, and returns a pointer to the newly created object. Use this to set further attributes on the event. Call Release on the returned event when it is no longer required. It will remain in the stored list until explicitly removed.

4.5.2 Event

Declaration: HRESULT Event([in]long i_Index, [out, retval]Event ** retval)

Return type: Event — Event retrieved

Parameters: long i_Index — Index of event to get

Description: Obtains the requested event from the stored list. Call Release on the returned event when it is no longer required.

4.5.3 EventContext

Declaration: HRESULT EventContext([in]long i_Index, [out, retval]EventContext ** retval)

Return type:	EventContext	Event context retrieved
Parameters:	long i_Index	Index of event to get

Description: Obtains the requested event context from the stored list. Call Release on the returned event context when it is no longer required.

4.5.4 EventContext Count

Declaration: HRESULT EventContextCount([out, retval]long* retval)

Return type:	long	Number of event contexts
Parameters:	NONE	

Description: Obtains the number of event contexts in the store.

4.5.5 EventCount

Declaration: HRESULT EventCount([out, retval]long* retval)

Return type:	long	Number of events
Parameters:	NONE	

Description: Obtains the number of events in the store.

4.5.6 RemoveEvent

Declaration: HRESULT RemoveEvent([in]Event * i_Event)

Return type:	NONE	
Parameters:	Event i_Event	Event to remove

Description: Removes the given event from the store.

4.5.7 RemoveEventAt

Declaration: HRESULT RemoveEventAt([in]long i_Index)

Return type:	NONE	
Parameters:	long i_Index	Index of event to remove

Description: Removes the event, at the specified index in the list, from the store.

4.6 ForcePlate

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents the stream of data from a single force plate. This interface is rarely implemented by plug-ins but, instead, plug-ins typically obtain a pointer to a force plate object through other means such as from a trial object (see Trial interface).
Methods:	1. AnalogToVideoRatio 2. CenterOfPressure 3. Corner 4. FirstSampleNum 8. ForceUnits 5. LastSampleNum 8. MomentUnits 8. PointUnits 6. Reaction 7. SampleRate
Properties:	

4.6.1 AnalogToVideo Ratio

Declaration: HRESULT AnalogToVideoRatio([out, retval]Ratio ** retval)

Return type:	Ratio	Ratio interface
Parameters:	NONE	

Description: Obtains the ratio of the analog to video capture rates for this force plate. Call Release on the returned object when it is no longer required.

4.6.2 CenterOfPressure

Declaration: HRESULT CenterOfPressure([in]long i_SampleIndex, [out, retval]Point **)

Return type:	Point	Centre of pressure position
Parameters:	long a_SampleIndex	Sample index to obtain data for

Description: Obtains the centre of pressure position of the force on the plate in laboratory co-ordinates. Call Release on the returned object when it is no longer required.

4.6.3 Corners

Declaration: HRESULT Corner([in]short i_Corner. [out, retval]Point ** retval)

Return type:	Point	Position of this corner
Parameters:	short i_Corner	Corner to get the position for

Description: Returns the 3-dimensional co-ordinates of the corners of the force plate. There are four corners, ordered in a clockwise fashion and the corner index is 0 based. Call Release on the returned object when it is no longer required.

4.6.4 FirstSampleNum

Declaration: HRESULT FirstSampleNum([out, retval]long* retval)

Return type:	long	Sample number
Parameters:	NONE	

Description: Returns the first sample index for which data is available on this force plate.

4.6.5 ForceUnits

Declaration: HRESULT ForceUnits([out, retval]BSTR* retval)

Return type:	BSTR	Force units
Parameters:	NONE	

Description: Returns the force units for the force plate in the form of a text string.

4.6.6 GetCentreOf Pressures

Declaration: HRESULT GetCenterOfPressures([in]long i_FirstSample, [in]long i_LastSample, [out, retval]VARIANT* retval)

Return type:	VARIANT	3 x N array of co-ordinates
Parameters:	long i_FirstSample	Range begin field number
	long i_LastSample	Range end field number

Description: Obtains the center of pressure for the force plate over the specified one-based sample range (inclusive of last sample) as a 3 x N array.

4.6.7 GetForces

Declaration: HRESULT GetForces([in]long i_FirstSample, [in]long i_LastSample, [out, retval]VARIANT* retval)

Return type:	VARIANT	3 x N array of co-ordinates
Parameters:	long i_FirstSample	Range begin field number
	long i_LastSample	Range end field number

Description: Obtains the force for the force plate over the specified one-based sample range (inclusive of last sample) as a 3 x N array.

4.6.8 GetMoments

Declaration: HRESULT GetMoments([in]long i_FirstSample, [in]long i_LastSample, [out, retval]VARIANT* retval)

Return type:	VARIANT	3 x N array of co-ordinates
Parameters:	long i_FirstSample	Range begin field number
	long i_LastSample	Range end field number

Description: Obtains the moment for the force plate over the specified one-based sample range (inclusive of last sample) as a 3 x N array.

4.6.9 LastSampleNum

Declaration: HRESULT LastSampleNum([out, retval]long* retval)

Return type:	long	Sample number
Parameters:	NONE	

Description: Returns the last, inclusive sample index for which data is available on this force plate.

4.6.10 MomentUnits

Declaration: HRESULT MomentUnits([out, retval]BSTR* retval)

Return type:	BSTR	Moment units
Parameters:	NONE	

Description: Returns the moment units for the force plate in the form of a text string.

4.6.11 PointUnits

Declaration: HRESULT ForceUnits([out, retval]BSTR* retval)

Return type:	BSTR	Point units
Parameters:	NONE	

Description: Returns the 3-dimensional co-ordinate units for the force plate in the form of a text string.

4.6.12 Reaction

Declaration: HRESULT Reaction([in]long i_SampleIndex, [out, retval]Reaction ** retval)

Return type:	Reaction	Force plate reaction
Parameters:	long a_SampleIndex	Sample index to obtain data for

Description: Obtains the force, moment and moment reference point of the force plate in laboratory co-ordinate space. Call Release on the returned object when it is no longer required.

4.6.13 SampleRate

Declaration: HRESULT SampleRate([out, retval]float* retval)

Return type:	float	Sample rate for this force plate
Parameters:	NONE	

Description: Returns the sample rate of the data for this force plate, in Hz.

4.7 ParameterStore

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Used for storing groups of named parameters of different types, including arrays of those types. Groups are also known as sections which is how they are referred to in this interface. Trials may contain a parameter store component which gives direct access to data contained in a C3D file. This interface may also be used to map to data stored in an INI file.
Methods:	1. ContainsParameter 2. ContainsSection 3. CreateBooleanArray 4. CreateColourArray 5. CreateDateTimeArray 6. CreateIntegerArray 7. CreateRealArray 8. CreateTextArray 9. DeleteParameter 10. DeleteSection 11. GetBoolean 12. GetColour 13. GetDateTime 14. GetDimensionCount 15. GetDimension 16. GetInteger 17. GetReal 18. GetText 19. GetTime 20. SectionCount 21. SectionName 22. SetBoolean 23. SetColour 24. SetDateTime 25. SetDefaultTextLength 26. SetInteger 27. SetReal 28. SetText 29. SetTime
Properties:	

4.7.1 Contains Parameter

Declaration: HRESULT ContainsParameter([in]BSTR i_Section, [in]BSTR i_Parameter, [out, retval]BOOL* retval)

Return type: BOOL — Boolean for parameter existing

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name

Description: Returns TRUE if the parameter store contains the given parameter in the given section, else FALSE.

4.7.2 ContainsSection

Declaration: HRESULT ContainsSection([in]BSTR i_Section, [out, retval]BOOL* retval)

Return type: BOOL — Boolean for section existing

Parameters:

BSTR i_Section	Section name

Description: Returns S_TRUE if the parameter store contains the given section, else S_FALSE.

4.7.3 CreateBoolean Array

Declaration: HRESULT CreateBooleanArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]VARIANT* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
VARIANT* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of boolean (BOOL) values.

4.7.4 CreateColour Array

Declaration: HRESULT CreateColourArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]short* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
short* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of colour values.

Colours are stored as RGB triplets so the created array will actually have an extra dimension of size 3 but this should not be accounted for in the dimension parameters passed.

4.7.5 CreateDateTimeArray

Declaration: HRESULT CreateDateTimeArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]short* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
short* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of date and time values.

Date and time values are stored as Year, Month, Day, Hour, Minute and Second sets so the created array will actually have an extra dimension of size 6 but this should not be accounted for in the dimension parameters passed.

4.7.6 CreateIntegerArray

Declaration: HRESULT CreateIntegerArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]short* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
short* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of integer values.

4.7.7 CreateRealArray

Declaration: HRESULT CreateRealArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]short* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
short* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of single precision real (FLOAT) values.

4.7.8 CreateTextArray

Declaration: HRESULT CreateTextArray([in]BSTR i_Section, [in]BSTR i_Parameter, [in]short i_Dimensions, [in]short* i_pDimensionArray)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Dimensions	Number of array dimensions
short* i_pDimensionArray	Array of dimension values

Description: Creates a parameter in the given section with the given name to store an array of text characters.

4.7.9 Delete Parameter

Declaration: HRESULT DeleteParameter([in]BSTR i_pSection, [in]BSTR i_Parameter)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name

Description: Deletes the given parameter from the given section in the parameter store. There is no error if the parameter or section does not exist.

4.7.10 DeleteSection

Declaration: HRESULT DeleteSection([in]BSTR i_Section)

Return type: NONE

Parameters:

BSTR i_Section	Section name

Description: Deletes the given section from the parameter store. There is no error if the section does not exist.

4.7.11 GetBoolean

Declaration: HRESULT Boolean([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, retval]BOOL* retval)

Return type:	BOOL	Parameter value
Parameters:	BSTR i_Section	Section name
	BSTR i_Parameter	Parameter name
	long i_Index	Value index

Description: Obtains the requested parameter value as a boolean, performing a type conversion if necessary. For integer and real parameters, a non-zero value indicates TRUE and a zero value indicates FALSE. For text parameters, any of “True”, “On”, “1” or “Yes” (case in-sensitive) are considered as TRUE. Any other value is considered to represent FALSE.

The zero-based value index must be calculated by transforming the parameter dimensions into a single dimensional array where the first dimension in a multi-dimensional array represents the least significant, i.e. the one that changes most often. For example, in the two-dimensional, 3 by 4 ([3][4]) array, the values are indexed [0..2][0..3]. The storage order for terms is [0][0], [1][0], [2][0], [0][1], etc. So the index for element [1][2] will be 2 * 4 + 1 = 9.

For single value parameters, the index will always be 0.

You can determine the dimensionality of a parameter using the GetDimensions method.

4.7.12 GetColour

Declaration: HRESULT Colour([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, UINT a_Index, [out, retval]Colour ** retval)

Return type:	Colour	Parameter value
Parameters:	BSTR i_Section	Section name
	BSTR i_Parameter	Parameter name
	long i_Index	Value index

Description: Obtains the requested colour parameter as a Colour interface object. Call Release on the returned object when it is no longer required.

See Boolean for a description of how to calculate the value index.

###

At the time of writing, this function is not implemented in Workstation software.

###

4.7.13 GetDateTime

Declaration: HRESULT DateTime([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, retval]DATE* retval)

Return type:

DATE	Parameter value

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
DATE i_Value	New parameter value

Description: Obtains the requested date and time parameter as a standard COM DATE type. This is a floating point number which represents days as whole number increments starting from midnight 30th December 1899.

See Boolean for a description of how to calculate the value index. Note also that the dimensionality will be one higher than expected since the date values are stored as an array of three integers.

###

At the time of writing, this function is not implemented in Workstation software.

###

4.7.14 GetDimension Count

Declaration: HRESULT GetDimensionCount([in]BSTR i_Section, [in]BSTR i_Parameter, [out, retval]short* retval)

Return type:

short	Number of array dimension

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name

Description: Gets the number of dimensions of the specified parameter.

4.7.15 GetDimension

Declaration: HRESULT Dimension([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, retval]short* retval)

Return type:

short	Size of array dimension

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
short i_Index	Dimension to obtain size for

Description: Gets the size of the indexed dimension of the specified parameter.

4.7.16 GetInteger

Declaration: HRESULT Integer([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, retval]long* retval)

Return type:	long	Parameter value
Parameters:	BSTR i_Section	Section name
	BSTR i_Parameter	Parameter name
	long i_Index	Value index

Description: Obtains the requested integer parameter as a long.

See Boolean for a description of how to calculate the value index.

4.7.17 GetReal

Declaration: HRESULT Real([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, retval]double* retval)

Return type:	double	Parameter value
Parameters:	BSTR i_Section	Section name
	BSTR i_Parameter	Parameter name
	long i_Index	Value index

Description: Obtains the requested real parameter as a double.

See Boolean for a description of how to calculate the value index.

4.7.18 GetText

Declaration: [propput] HRESULT Text([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]BSTR i_Value)

[propget] HRESULT Text([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [out, reval]BSTR* retval)

Return type:	BSTR	Parameter value
Parameters:	BSTR i_Section	Section name
	BSTR i_Parameter	Parameter name
	long i_Index	Value index

Description: Obtains the requested text parameter as a text string.

See Boolean for a description of how to calculate the value index.

4.7.19 SectionCount

Declaration: HRESULT SectionCount([out, retval]long* retval)

Return type:	long	Section count
Parameters:	NONE	

Description: Obtains the number of sections in the parameter store.

4.7.20 SectionName

Declaration: HRESULT SectionName([in]long i_Index, [out, retval]BSTR* retval)

Return type: BSTR — Section name

Parameters:

long i_Index	Section index

Description: Obtains the name of the section in the form of a text string.

4.7.21 SetBoolean

Declaration: HRESULT Boolean([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]BOOL i_Value)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
BOOL i_Value	New parameter value

Description: Sets the requested parameter value as a boolean.

The zero-based value index must be calculated by transforming the parameter dimensions into a single dimensional array where the first dimension in a multi-dimensional array represents the least significant, i.e. the one that changes most often. For example, in the two-dimensional, 3 by 4 ([3][4]) array, the values are indexed [0..2][0..3]. The storage order for terms is [0][0], [1][0], [2][0], [0][1], etc. So the index for element [1][2] will be 2 * 4 + 1 = 9.

For single value parameters, the index will always be 0.

You can determine the dimensionality of a parameter using the GetDimensions method.

4.7.22 SetColour

Declaration: HRESULT Colour([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, UINT a_Index, [in]Colour * i_Colour)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
Colour i_Value	New parameter value

Description: ###

At the time of writing, this function is not implemented in Workstation software.

###

4.7.23 SetDateTime

Declaration: HRESULT DateTime([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]DATE i_Date)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
DATE i_Value	New parameter value

Description: ###

At the time of writing, this function is not implemented in Workstation software.

###

4.7.24 SetDefault TextLength

Declaration: HRESULT SetDefaultTextLength([in]long retval)

Return type: NONE

Parameters:

long i_Length	Default length

Description: Sets the default length of the text strings stored in the parameter store.

4.7.25 SetInteger

Declaration: HRESULT Integer([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]long i_Value)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
long i_Value	New parameter value

Description: Sets the requested integer parameter as a long.

See Boolean for a description of how to calculate the value index.

4.7.26 SetReal

Declaration: HRESULT Real([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]double i_Value)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
double i_Value	New parameter value

Description: Sets the requested real parameter as a double.

See Boolean for a description of how to calculate the value index.

4.7.27 SetText

Declaration: HRESULT Text([in]BSTR i_Section, [in]BSTR i_Parameter, [in]long i_Index, [in]BSTR i_Value)

Return type: NONE

Parameters:

BSTR i_Section	Section name
BSTR i_Parameter	Parameter name
long i_Index	Value index
BSTR i_Value	New parameter value

Description: Sets the requested text parameter as a text string.

See Boolean for a description of how to calculate the value index.

4.8 PECS

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents the base level document from which all other interfaces can be obtained hierarchically. Although the Trial, Processor and RefCounter interfaces are exposed as properties, these should never be set by a client process. These interfaces are set internally by Workstation or a Workstation Plug-In (VPI) by instantiating concrete objects and passing their IDispatch interfaces:
Methods:	1. CapabilityFlags 2. ConnectionCount 3. Execute 4. Modeller 5. Refresh 6. SupportsCapabilityAll 7. SupportsCapabilitySome
Properties:	8. Trial 9. Processor 10. RefCounter

4.8.1 CapabilityFlags

Declaration: HRESULT CapabilityFlags([in]long i_FlagSet, [out, retval]DWORD* retval)

Return type:	DWORD	Capability mask
Parameters:	long i_FlagSet	Flag to check for capability

Description: Obtains the bit mask for Workstation capabilities based on the index flag.

4.8.2 ConnectionCount

Declaration: HRESULT ConnectionCount([out, retval]long*)

Return type:	long	Connection count
Parameters:	NONE	

Description: Gets the number of processes currently holding a PECS interface. Calling Release on any PECS interface will cause this to reduce by 1. The server should automatically close down when this count reaches 0.

4.8.3 Execute

Declaration: STDMETHOD(Execute)(THIS_ LPCSTR a_pInstruction, LPCSTR a_pParameters)

Return type: NONE

Parameters:

LPCSTR a_pInstruction	in: Instruction to execute
LPCSTR a_pParameters	in: Parameters to instruction

Description: Executes the given instruction on the application.

The parameters must be supplied in textual form as a semicolon (';') separated list. Leading and trailing white space (space or tab characters) will be ignored for each parameter. Textual string parameters may optionally be enclosed in quotes (' " ') to allow for the inclusion of semicolon characters or leading and trailing white space in the string.

The return value will depend on the particular instruction but typically, NOERROR will be returned for success or E_FAIL will be returned if the instruction could not be executed.

The following instructions are supported in Vicon Workstation:

Version	Instruction	Parameters
3.0	CheckLicence	ProgramID;VersionID;FamilyID

4.8.4 Refresh

Declaration: HRESULT Refresh()

Return type: NONE

Parameters: NONE

Description: Causes the PECS to refresh its view.

4.8.5 Supports CapabilityAll

Declaration: HRESULT SupportsCapabilityAll([in]long i_FlagSet, [in]DWORD i_FlagMask, [out, retval]BOOL* retval)

Return type:

BOOL	Supports all flag

Parameters:

long i_FlagSet	Flag to check for capability
DWORD i_FlagMask	Capability mask

Description: Checks the given bit mask against the Workstation capability bit mask based on the index flag.

The return value will by TRUE only if all of the bits set in the mask match those in the Workstation capability mask. Otherwise, the function returns FALSE.

4.8.6 SupportsCapabilitySome

Declaration: HRESULT SupportsCapabilityAll([in]long i_FlagSet, [in]DWORD i_FlagMask, [out, retval]BOOL* retval)

Return type:	BOOL	Supports some flag
Parameters:	long i_FlagSet	Flag to check for capability
	DWORD i_FlagMask	Capability mask

Description: Checks the given bit mask against the Workstation capability bit mask based on the index flag.

The return value will by TRUE if any of the bits set in the mask match those in the Workstation capability mask. The function returns FALSE only if none of the set bits are matched.

4.8.7 Processor

Declaration: [propput] HRESULT Processor([in]Processor * i_pProcessor)
[propget] HRESULT Processor([out, retval]Processor ** retval)

Return type:	Processor	Processor interface
Parameters:	Processor i_pProcessor	New processor interface

Description: Sets and Gets the currently attached Processor interface. Call Release on the object when it is no longer required.

4.8.8 RefCounter

Declaration: [propput] HRESULT RefCounter([in]RefCounter * i_pRefCounter)
[propget] HRESULT RefCounter([out, retval]RefCounter ** retval)

Return type:	RefCounter	Trial interface
Parameters:	RefCounter i_pTrial	New trial interface

Description: Sets and Gets the currently attached RefCounter interface. Call Release on the object when it is no longer required.

4.8.9 Trial

Declaration: [propput] HRESULT Trial([in]Trial* i_pTrial)
[propget] HRESULT Trial([out, retval]Trial** retval)

Return type:	Trial	Trial interface
Parameters:	Trial i_pTrial	New trial interface

Description: Sets and Gets the currently attached Trial interface. Call Release on the object when it is no longer required.

4.9 Processor

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	An interface to the object representing the processor in pipeline processing. This simple interface is provided to allow feedback to users during processing.
Methods:	1. Log 2. SetStatusMessage
Properties:	

4.9.1 Log

Declaration: HRESULT Log([in]BSTR i_Text, [in]BOOL i_OnNewLine)

Return type: NONE

Parameters:

BSTR i_Text	Text to log
BOOL i_OnNewLine	Force new line or not

Description: Logs an event and associates the given text with it. The results in an entry being made to the log file and also to the processing log window.

If i_OnNewLine is TRUE (non-zero) then the text is forced onto a new line in the log with a time stamp automatically added to the start of the text.

Typically, this is used by processes such that a “Working… ” message is logged on a new line and when processing is complete, a further “Results” string is logged but not on a new line producing: “09:37:05 - Working… Results” in the log.

4.9.2 SetStatus Message

Declaration: HRESULT SetStatusMessage([in]BSTR i_Text)

Return type: NONE

Parameters:

BSTR i_Text	in: Status text

Description: Outputs the given text in the status bar of the applications main window. This is useful during length pipeline processing to give feedback of progress to users.

4.10 Subject

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents a measured subject, usually human. This provides information necessary to distinguish the data between subjects in a multiple subject trial. This interface is rarely implemented by plug-ins but, instead, plug-ins typically obtain a pointer to an analog channel object through other means such as from a trial object (see IVTrial interface).
Methods:	1. DisplaySetName 2. LabelPrefix 3. MarkerSetName 4. ModelName 5. ModelParamsName 6. Name 7. LabelPrefixUsage 8. MarkerSet
Properties:	

4.10.1 DisplaySetName

Declaration: HRESULT DisplaySetName([out, retval]BSTR* retval)

Return type: BSTR — Display name

Parameters: NONE

Description: Returns the name of the active display set within the marker set in the form of a text string.

4.10.2 LabelPrefix

Declaration: HRESULT LabelPrefix([out, retval]BSTR* retval)

Return type: BSTR — Label prefix

Parameters: NONE

Description: Returns the subject label prefix in the form of a text string. This prefix is used on trajectories to associate the trajectory wit the subject.

4.10.3 MarkerSetName

Declaration: HRESULT MarkerSetName([out, retval]BSTR* retval)

Return type: BSTR — Marker set name

Parameters: NONE

Description: Returns the name of the marker set file in the form of a text string. The name may or may not include a full path and file extension.

4.10.4 ModelName

Declaration: HRESULT ModelName([out, retval]BSTR* retval)

Return type: BSTR — Model name

Parameters: NONE

Description: Returns the name of the model file in the form of a text string. The name may or may not include a full path and file extension.

4.10.5 ModelParams Name

Declaration: HRESULT ModelParamsName([out, retval]BSTR* retval)

Return type: BSTR — Model params name

Parameters: NONE

Description: Returns the name of the model parameters file in the form of a text string. The name may or may not include a full path and file extension.

4.10.6 Name

Declaration: HRESULT Name([out, retval]BSTR* retval)

Return type: BSTR — Subject name

Parameters: NONE

Description: Returns the subject name in the form of a text string.

4.10.7 SetLabel PrefixUsage

Declaration: HRESULT SetLabelPrefixUsage([in]BOOL i_UsesPrefix)

Return type: NONE

Parameters: BOOL i_UsesPrefix — Whether prefix is active or not

Description: Sets whether a (trajectory) label prefix is active for the subject or not. If the value is TRUE (non-zero), then prefixes are activated and the prefix is formed from the subject name followed by a colon (':') character. If the value is FALSE (zero) then the prefix is set to nothing for the subject.

Note that this function does not affect the labels on any trajectories, whether associated with the subject or not.

4.10.8 SetMarkerSet

Declaration: HRESULT SetMarkerSet([in]BSTR i_MKRFilename)

Return type: NONE

Parameters:

BSTR i_MKRFilename	New marker set filename

Description: Sets the marker set (file) for the subject. The display set is reset to the first one available in the new file.

Note that this function does not affect the labels on any trajectories, whether associated with the subject or not.

4.11 Trajectory

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents the 3-dimensional trajectory of a point through space, measured or virtual. May also be used to represent other 3-dimensional quantities. This interface is rarely implemented by plug-ins but, instead, plug-ins typically obtain a pointer to a trajectory object through other means such as from a trial object (see IVTrial interface). Note that, unless stated otherwise, changes to trajectories are not reflected in the user interface until a call is made to RefreshChanges on the Trial object. This is to allow multiple changes to occur with only a single refresh for maximum performance.
Methods:	1. ClearSelectedPoints 2. ClearTrajectory 3. FindGapAt 4. FindGapBackward 5. FindGapForward 6. FirstValidFieldNum 7. GetPoint 8. GetPoints 9. GetPointCameras 10. HasEmptyLabel 11. Hide 12. InvalidatePoint 13. IsVisible 14. LastValidFieldNum 15. MatchesLabel 16. MatchesLabelPrefix 17. PointCamera 18. PointCameraCount 19. SampleRate 20. SelectedPointCount 21. SelectedPointField 22. SelectPoint 23. SetPoint 24. SetPoints 25. Show 26. Units

Properties:	27. Colour 28. Label 29. MarkerSize 30. TypeGroup

4.11.1 ClearSelected Points

Declaration: HRESULT ClearSelectedPoints()

Return type: NONE

Parameters: NONE

Description: Removes all point selections from the trajectory and leaves it in a deselected state.

4.11.2 ClearTrajectory

Declaration: HRESULT ClearTrajectory()

Return type: NONE

Parameters: NONE

Description: Empties the trajectory of all points and leaves it in a deselected state.

4.11.3 FindGapAt

Declaration: HRESULT FindGapAt([in]long i_SearchField, [out, retval]Range ** retval)

Return type: Range — Range of the found gap

Parameters: long i_SearchField — Field to effect the search at

Description: Tests whether there is a gap at a given field in the trajectory without searching elsewhere.

If the point at the specified field is invalid, the start and end of the gap are determined and the returned Range object will have a size of greater than zero. Otherwise, the returned range will have zero size, to signify that there is no gap at that field. Call Release on the returned object when it is no longer required.

Note that the start and end fields represent the bounds of the gap and, by definition, the points at those fields will both be invalid.

4.11.4 FindGap Backward

Declaration: HRESULT FindGapBackward([in]long i_SearchFrom, [in]long i_IgnoreValidLen, [out, retval]Range ** retval)

Return type:	Range	Range of the found gap
Parameters:	long i_SearchFrom	Field number to start search from
	long i_IgnoreValidLen	Valid fields ignored within gap

Description: Searches backward through the trajectory, starting from the specified field, looking for gaps consisting of invalid points. Groups of consecutive valid points shorter or equal to IgnoreValidLen will be treated as part of the gap. Set this parameter to zero to find true gaps where all points within are invalid.

If a gap is found, the returned Range object will have a size of greater than zero. Otherwise, the returned range will have zero size. Call Release on the returned object when it is no longer required.

Note that the start and end fields represent the bounds of the gap and, by definition, the points at those fields will both be invalid.

To find the previous gap, start the search from the start field of the found gap minus one.

4.11.5 FindGap Forward

Declaration: HRESULT FindGapForward([in]long i_SearchFrom, [in]long i_IgnoreValidLen, [out, retval]Range ** retval)

Return type:	Range	Range of the found gap
Parameters:	long i_SearchFrom	Field number to start search from
	long i_IgnoreValidLen	Valid fields ignored within gap

Description: Searches forward through the trajectory, starting from the specified field, looking for gaps consisting of invalid points. Groups of consecutive valid points shorter or equal to IgnoreValidLen will be treated as part of the gap. Set this parameter to zero to find true gaps where all points within are invalid.

If a gap is found, the returned Range object will have a size of greater than zero. Otherwise, the returned range will have zero size. Call Release on the returned object when it is no longer required.

Note that the start and end fields represent the bounds of the gap and, by definition, the points at those fields will both be invalid.

To find the next gap, start the search from the end field of the found gap plus one.

4.11.6 FirstValidFieldNum

Declaration: HRESULT FirstValidFieldNum([out, retval]long* retval)

Return type:	long	First valid field number
Parameters:	NONE	

Description: Returns the one-based field number of the first valid point in the trajectory. This will be zero if there are no valid points in the trajectory.

4.11.7 GetPoint

Declaration: HRESULT GetPoint([in]long i_Field, [out, retval]Reconstruction ** retval)

Return type:	Reconstruction	Reconstructed
Parameters:	long i_Field	Field number

Description: Obtains the co-ordinates of the trajectory at the specified one-based field number as a Reconstruction interface. Call Release on the returned object when it is no longer required.

4.11.8 GetPoints

Declaration: HRESULT GetPoints([in]long i_Begin, [in]long i_End, [out, retval]VARIANT* retval)

Return type:	VARIANT	3 x N array of co-ordinates
Parameters:	long i_Begin	Range begin field number
	long i_End	Range end field number

Description: Obtains the co-ordinates of the trajectory over the specified one-based field range (inclusive of end frame) as a 3 x N array.

4.11.9 GetPointCameras

Declaration: HRESULT GetPointCameras([in]long i_Begin, [in]long i_End, [out, retval]VARIANT* retval)

Return type:	VARIANT	32 x N array of contributions
Parameters:	long i_Begin	Range begin field number
	long i_End	Range end field number

Description: Obtains the camera contributions over the specified one-based field range (inclusive of end frame) as a 32 x N array. A 1 in the cell (i, j) indicates that the i^{th} camera contributed to the j^{th} frame.

4.11.10 HasEmptyLabel

Declaration: HRESULT HasEmptyLabel([out, retval]BOOL* retval)

Return type: BOOL — Empty label flag

Parameters: NONE

Description: Returns TRUE if the trajectory has a blank label and is therefore unlabelled, otherwise FALSE.

4.11.11 Hide

Declaration: HRESULT Hide()

Return type: NONE

Parameters: NONE

Description: Causes the trajectory to be made invisible in the user interface Workspace.

4.11.12 InvalidatePoint

Declaration: HRESULT InvalidatePoint([in]long i_Field)

Return type: NONE

Parameters: long i_Field — Field to invalidate

Description: Invalidates the point in the trajectory at the specified field. This effectively means that the point is missing.

4.11.13 IsVisible

Declaration: HRESULT IsVisible([out, retval]BOOL* retval)

Return type: BOOL — Is visible flag

Parameters: NONE

Description: Returns TRUE if the trajectory is visible in the user interface Workspace, else FALSE.

4.11.14 LastValidFieldNum

Declaration: LastValidFieldNum([out, retval]long* retval)

Return type: long — last valid field number

Parameters: NONE

Description: Returns the one-based field number of the last valid point in the trajectory. This will be zero if there are no valid points in the trajectory.

4.11.15 MatchesLabel

Declaration: HRESULT MatchesLabel([in]BSTR i_Label, [out, retval]BOOL* retval)

Return type:	BOOL	Label matches flag
Parameters:	BSTR i_Label	Label to compare

Description: Convenient way of directly checking for a match against a given label rather than using GetLabel and checking that manually. Note that subject prefixes and other modifiers must be included in the label given.

Returns TRUE if the label matches, otherwise FALSE.

4.11.16 MatchesLabel Prefix

Declaration: HRESULT MatchesLabelPrefix([in]BSTR i_Label, [out, retval]BOOL* retval)

Return type:	BOOL	Label prefix matches flag
Parameters:	BSTR i_Label	Label prefix to compare

Description: Checks whether the trajectory matches the given subject prefix. Note that the prefix separator must be included in the prefix given.

Returns TRUE if the prefix matches, otherwise FALSE.

4.11.17 PointCamera

Declaration: HRESULT PointCamera([in]long i_Field, [in]long i_Index,
[out, retval]long* retval)

Return type:	long	Camera number
Parameters:	long i_Field	Field number
	long i_Index	Camera index

Description: **###**

At the time of writing, this function is not implemented in Workstation software.

###

4.11.18 PointCamera Count

Declaration: HRESULT PointCameraCount([in]long i_Field, [out, retval]long* retval)

Return type:	long	Camera count
Parameters:	long i_Field	Field number

Description: **###**

At the time of writing, this function is not implemented in Workstation software.

###

4.11.19 SampleRate

Declaration: HRESULT SampleRate([out, retval]double* retval)

Return type: double — Sample rate

Parameters: NONE

Description: Returns the sample rate of the trajectory data in Hz.

4.11.20 SelectedPoint Count

Declaration: HRESULT SelectedPointCount([out, retval]long* retval)

Return type: long — Count

Parameters: NONE

Description: Obtains the number of selected points in the trajectory. This will typically be 0 if the trajectory is not selected , 1 if the trajectory is selected at a single point or 2 if a range of points has been selected on the trajectory. Higher numbers could indicate multiple selection of individual points but the meaning is open to interpretation depending on desired functionality.

4.11.21 SelectedPoint Field

Declaration: HRESULT SelectedPointField([in]long i_Index, [out, retval]long* retval)

Return type: long — Field number

Parameters: long i_Index — Selection index

Description: Obtains the one-based field number of the Nth selected point on the trajectory, where N is given by the selection index and must lie within the bounds determined by a call to SelectedPointCount.

4.11.22 Units

Declaration: HRESULT Units([out, retval]BSTR* retval)

Return type: BSTR — Units

Parameters: NONE

Description: Returns the trajectory point units in the form of a text string.

4.11.23 SelectPoint

Declaration: HRESULT SelectPoint([in]long i_Field)

Return type: NONE

Parameters: long i_Field — Field number to select

Description: Selects the trajectory point at the given one-based field number. The call may fail if the maximum number of points (typically two) are already selected in the trajectory.

4.11.24 SetPoint

Declaration: HRESULT GetPoint([in]long i_Field, [in]Reconstruction * i_Point)

Return type: NONE

Parameters:

long i_Field	Field number
Reconstruction* i_Point	New reconstructed point

Description: Sets the co-ordinates of the trajectory at the specified one-based field number as a Reconstruction interface. Call Release on the returned object when it is no longer required.

4.11.25 SetPoints

Declaration: HRESULT SetPoints([in]long i_Begin, [in]long i_End, [in]VARIANT* i_Points)

Return type: NONE

Parameters:

long i_Begin	Range begin field number
long i_End	Range end field number
VARIANT* i_Points	3 x N array of co-ordinates

Description: Sets the co-ordinates of the trajectory over the specified one-based field range (inclusive of end frame) as 3 x N matrix of co-ordinates. If the matrix passed in does not have the same number of columns as there are fields in the specified range, the server throws an exception.

4.11.26 Show

Declaration: HRESULT Show()

Return type: NONE

Parameters: NONE

Description: Makes the trajectory visible in the user interface Workspace.

4.11.27 Colour

Declaration: [propput] HRESULT Colour([in]Colour * i_Colour)

[propget] HRESULT Colour([out, retval]Colour ** retval)

Return type: Colour — Trajectory colour

Parameters: Colour i_Colour — New trajectory colour

Description: ###

At the time of writing, this function is not implemented in Workstation software.

###

4.11.28 Label

Declaration: [propput] HRESULT Label([in]BSTR i_Label)
[propget] HRESULT Label([out, retval]BSTR* retval)

Return type:	BSTR	Trajectory label
Parameters:	BSTR i_Label	New trajectory label

Description: Sets and gets the trajectory label in the form of a text string. Any subject prefix or other modifier must be provided as part of the label.

4.11.29 MarkerSize

Declaration: [propput] HRESULT MarkerSize([in]double i_MarkerSize)
[propget] HRESULT MarkerSize([out, retval]double retval)

Return type:	double	Marker size
Parameters:	double i_MarkerSize	New marker size

Description: ###
At the time of writing, this function is not implemented in Workstation software.
###

4.11.30 TypeGroup

Declaration: [propput] HRESULT TypeGroup([in]BSTR i_TypeGroup)
[propget] HRESULT TypeGroup([out]BSTR* i_TypeGroup)

Return type:	BSTR	Trajectory type name
Parameters:	BSTR i_TypeGroup	New trajectory type name

Description: Sets and gets the 'type' of the trajectory in the form of a text string. Different types may be used to store vector information in a trajectory other than measured point data. For example, the following types are often used: "FORCE", "MOMENT", and "POWER".

4.12 Trial

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	Represents a trial containing subjects, reconstructed trajectories, analog data, force plate data and other items. This interface contains functions for getting at these items, represented as separate objects with their own interfaces, so that they may be manipulated. This interface is rarely implemented by plug-ins but, instead, a pointer to a trial object is typically passed to plug-in functions (for example, see the IVProcess interface).
Methods:	1. AnalogBaseRate 2. AnalogChannel 3. AnalogChannelCount 4. CreateTrajectory 5. DataPath 6. DateAndTime 7. DefaultMarkerSize 8. DeleteAllTrajectories 9. DeleteAllUnlabelledTrajectories 10. DeleteTrajectory 11. Description 12. EclipseNode 13. EventStore 14. Execute 15. FindAnalogChannel 16. FindSubject 17. FindSubjectByPrefix 18. FindTrajectory 19. FindTrajectoryIndex 20. FirstValidTrajectoryFieldNum 21. ForcePlate 22. ForcePlateCount 23. IsStatic 24. LastValidTrajectoryFieldNum 25. Notes 26. ParameterStore 27. ReferenceName 28. RefreshChanges 29. RegisterTrajectoryTypeGroup 30. SetDefaultMarkerSize

	31. SetLabelPrefixUsage 32. SetStaticStatus 33. SetTrajectoryTypeGroupUnits 34. Subject 35. SubjectCount 36. Trajectory 37. TrajectoryCount 38. TrajectoryTypeGroupCount 39. TrajectoryTypeGroupName 40. TrajectoryTypeGroupUnits 41. Type 42. UsesLabelPrefixes 43. VideoFieldCount 44. VideoRate
Properties:	

4.12.1 AnalogBaseRate

Declaration: HRESULT AnalogBaseRate([out, retval]double* retval)

Return type: double — Analog base rate

Parameters: NONE

Description: Returns the base sample rate of the trial analog data in Hz. Individual analog channels may be sampled at a sub-multiple of this rate according to the base rate divider for that channel.

4.12.2 AnalogChannel

Declaration: HRESULT AnalogChannel([in]long i_Index, [ot, retval]AnalogChannel ** retval)

Return type: AnalogChannel — Analog channel interface

Parameters: long i_Index — Zero-based channel index

Description: Obtains the specified analog channel as a pointer to an interface. Call Release on the object when it is no longer required.

If the requested analog channel index is out of range, the server throws an exception. Use AnalogChannelCount to get the number of analog channels present.

4.12.3 AnalogChannel Count

Declaration: HRESULT AnalogChannelCount([out, retval]long* retval)

Return type: long — Channel count

Parameters: NONE

Description: Obtains the number of analog channels in the trial.

4.12.4 CreateTrajectory

Declaration: HRESULT CreateTrajectory([out, retval]Trajectory ** retval)

Return type: Trajectory — Trajectory interface

Parameters: NONE

Description: Creates an empty, new trajectory in the trial, returning it as a Trajectory interface.
Call Release on the object when it is no longer required.

4.12.5 DataPath

Declaration: HRESULT DataPath([out, retval]BSTR* retval)

Return type: BSTR — Data path

Parameters: NONE

Description: Returns the trial data path in the form of a text string.

4.12.6 DateAndTime

Declaration: HRESULT DateAndTime([out, retval]DATE* retval)

Return type: DATE — Date and time

Parameters: NONE

Description: Obtains the date and approximate time that the trial was captured as a standard COM DATE type. This is a floating point number which represents days as whole number increments starting from midnight 30th December 1899.

4.12.7 DefaultMarker Size

Declaration: HRESULT DefaultMarkerSize([out, retval]double* retval)

Return type: double — Default marker size

Parameters: NONE

Description: Returns the default size for markers in millimetres (mm). This is the size assigned to new trajectories.

4.12.8 DeleteAll Trajectories

Declaration: HRESULT DeleteAllTrajectories()

Return type: NONE

Parameters: NONE

Description: Deletes all trajectories from the trial.

4.12.9 DeleteAll Unlabelled Trajectories

Declaration: HRESULT DeleteAllUnlabelledTrajectories()

Return type: NONE

Parameters: NONE

Description: Deletes all trajectories from the trial that do not have an assigned label.

4.12.10 DeleteTrajectory

Declaration: HRESULT DeleteTrajectory([in]long i_Index)

Return type: NONE

Parameters:

long i_Index	Trajectory index

Description: Deletes the trajectory at the given index (0 .. GetTrajectoryCount()-1). Note that this will change the index values of all trajectories with a higher index. When deleting multiple trajectories, either adjust the indexes as each trajectory is deleted, delete them in descending order of index so that they remain valid, or re-compute the index values each time, for example, using the FindTrajectoryIndex function.

4.12.11 Description

Declaration: HRESULT Description([out, retval]BSTR* retval)

Return type:

BSTR	Description

Parameters: NONE

Description: Returns the description of the trial in the form of a text string. This is the description that is typed into the Directory window.

4.12.12 EclipseNode

Declaration: HRESULT EclipseNode([out, retval]EclipseNode * retval)

Return type:

EclipseNode	Eclipse node interface

Parameters: NONE

Description: Obtains the eclipse node for this trial in the form of an EclipseNode interface. Call Release on this object when it is no longer required.

4.12.13 EventStore

Declaration: HRESULT EventStore([out, retval]EventStore * retval)

Return type:

EventStore	Data path

Parameters: NONE

Description: Obtains the event store for this trial in the form of an EventStore interface. Call Release on this object when it is no longer required.

4.12.14 Execute

Declaration: HRESULT Execute([in]BSTR i_Instruction, [in]BSTR i_Parameters, [out, retval]BOOL* retval)

Return type:	BOOL	Success flag
Parameters:	BSTR i_Instruction	Instruction to execute
	BSTR i_Parameters	Parameters to instruction

Description: Executes the given instruction on the trial and/or application.

The parameters must be supplied in textual form as a semicolon (';') separated list. Leading and trailing white space (space or tab characters) will be ignored for each parameter. Textual string parameters may optionally be enclosed in quotes (' " ') to allow the inclusion of semicolon characters or leading and trailing white space in the string.

The return value will depend on the particular instruction but typically, TRUE will be returned for success or FALSE will be returned if the instruction could not be executed.

The following instructions are supported in Vicon Workstation:

Version	Instruction	Parameters
3.0	CheckLicence	ProgramID;VersionID;FamilyID
3.0	ResetForcePlateOffsets	*(none)*
3.0	Defragment	*(none)*

4.12.15 FindAnalog Channel

Declaration: HRESULT FindAnalogChannel([in]BSTR a_pLabel, [out, retval]AnalogChannel ** retval)

Return type:	AnalogChannel	Analog channel interface
Parameters:	BSTR i_Label	Label of channel to find

Description: Searches for an analog channel in the trial that matches the given label, returning it as a pointer to an interface. In case of conflicts where more than one channel matches the label, this function returns the first one found.

If no matching analog channel can be found, the server throws an exception.

Call Release on the object when it is no longer required.

4.12.16 FindSubject

Declaration: HRESULT FindSubject([in]BSTR i_Name, [out, retval]Subject** retval)

Return type:	Subject	Subject interface
Parameters:	BSTR i_Name	Name of subject to find

Description: Searches for a subject in the trial that matches the given name, returning it as a pointer to a subject interface. Each subject must have a unique name in the trial. In case of conflicts, this function returns the first match found.

If no matching subject can be found, the server throws an exception.

Call Release on the object when it is no longer required.

4.12.17 FindSubjectByPrefix

Declaration: HRESULT FindSubjectByPrefix([in]BSTR i_Prefix, [out, retval]Subject** retval)

Return type:	Subject	Subject interface
Parameters:	BSTR i_Prefix	Prefix of subject to find

Description: Searches for a subject in the trial that matches the given label prefix, returning it as a pointer to a subject interface. Typically, each subject in a multiple-subject trial has a unique label prefix based on the subject name. In case of conflicts where this is not the case, this function returns the first match found.

If no matching subject can be found, the server throws an exception.

Call Release on the object when it is no longer required.

4.12.18 FindTrajectory

Declaration: HRESULT FindTrajectory([in]BSTR i_Label, [out, retval]Trajectory** retval)

Return type:	Trajectory	Trajectory interface
Parameters:	BSTR i_Label	Label of trajectory to find

Description: Searches for a trajectory in the trial that matches the given label, returning it as a pointer to a trajectory interface. The full label must be provided including any subject prefix or other modifier. In case of conflicts where more than one trajectory matches the label, this function returns the first one found.

If no matching trajectory can be found, the server throws an exception.

Call Release on the object when it is no longer required.

4.12.19 FindTrajectoryIndex

Declaration: HRESULT FindTrajectoryIndex([in]BSTR a_pLabel, [in]long i_FromIndex, [out, retval]long* retval)

Return type:	long	Found index
Parameters:	BSTR i_Label	Label to find
	long i_FromIndex	Index to start search at

Description: Searches from the given index for a trajectory in the trial that matches the given label, returning its index. The full label must be provided including any subject prefix or other modifier. In case of conflicts where more than one trajectory matches the label, this function returns the first one found.

Returns -1 if no matching trajectory can be found.

Note that index values may change as a result of creating or deleting other trajectories.

4.12.20 FirstValidTrajectoryFieldNum

Declaration: HRESULT FirstValidTrajectoryFieldNum([out, retval]long* retval)

Return type:	long	First valid field number
Parameters:	NONE	

Description: Obtains the inclusive field number of the first occurrence in the trial of valid trajectory data.

Returns -1 if no valid trajectory data exists. Note that this latter case may occur if the trajectory count is not zero but the trajectories that do exist contain no valid point data.

4.12.21 ForcePlate

Declaration: HRESULT ForcePlate([in]long i_Index, [out, retval]ForcePlate** retval)

Return type:	ForcePlate	Force plate interface
Parameters:	long i_Index	Zero-based force plate index

Description: Obtains the specified force plate as a pointer to an interface. Call Release on the object when it is no longer required.

4.12.22 ForcePlateCount

Declaration: HRESULT ForcePlateCount([out, retval]long* retval)

Return type:	long	Force plate count
Parameters:	NONE	

Description: Obtains the number of force plates defined for the trial.

4.12.23 IsStatic

Declaration: HRESULT IsStatic([out, retval]BOOL* retval)

Return type: BOOL — Is static flag

Parameters: NONE

Description: Returns S_TRUE if the trial has been marked as static, otherwise S_FALSE. Static is used to indicate that the measurement subject remained still for the whole duration of the trial.

4.12.24 LastValid TrajectoryFieldNum

Declaration: HRESULT LastValidTrajectoryFieldNum([out, retval]long* retval)

Return type: long — Last valid field number

Parameters: NONE

Description: Obtains the inclusive field number of the last occurrence in the trial of valid trajectory data.

Returns -1 if no valid trajectory data exists. Note that this latter case may occur if the trajectory count is not zero but the trajectories that do exist contain no valid point data.

4.12.25 Notes

Declaration: HRESULT Notes([out, retval]BSTR* retval)

Return type: BSTR — Notes

Parameters: NONE

Description: Returns the notes for the trial in the form of a text string. These are the notes that are typed into the Directory window.

4.12.26 Parameter Store

Declaration: HRESULT ParameterStrore([out, retval]ParameterStore* retval)

Return type: ParameterStore — Data path

Parameters: NONE

Description: Obtains the event store for this trial in the form of a ParameterStore interface. Call Release on this object when it is no longer required.

4.12.27 ReferenceName

Declaration: HRESULT ReferenceName([out, retval]BSTR* retval)

Return type: BSTR — Reference name

Parameters: NONE

Description: Returns the trial reference name in the form of a text string.

4.12.28 RefreshChanges

Declaration: HRESULT RefreshChanges()

Return type: NONE

Parameters: NONE

Description: Refreshes the user interface associated with the trial. Call this function after making changes to trial data that affect the user interface, for example, modifying or creating new trajectories.

If possible, call this function once after many changes rather than for each individual change.

4.12.29 Register TrajectoryTypeGroup

Declaration: HRESULT RegisterTrajectoryTypeGroup([in]BSTR i_Name, [in]BSTR i_SingularName)

Return type: NONE

Parameters:

BSTR i_Name	Plural name
BSTR i_SingularName	Singular name

Description: Registers the plural and singular names of a trajectory type group. If a group matching the plural name already exists then this registration will be ignored and the existing, original singular name will be maintained.

Trajectory type groups are a mechanism by which 3D information other than point trajectories can be stored. Examples include angles, forces and moments. Once a type group has been registered, individual trajectories may be assigned to the group using the SetTypeGroup function. This information can then be used by any process that needs to recognise the different data stored as trajectories.

4.12.30 SetDefault MarkerSize

Declaration: HRESULT SetDefaultMarkerSize([in]double I_MarkerSize)

Return type: NONE

Parameters:

double i_MarkerSize	Marker size

Description: ###

At the time of writing, this function is not implemented in Workstation software.

###

4.12.31 SetLabelPrefixUsage

Declaration: HRESULT SetLabelPrefixUsage([in]BOOL i_UsesPrefix)

Return type: NONE

Parameters:

BOOL i_UsesPrefix	New prefix usage flag

Description: Sets whether subject identifiers are used to prefix the labels of each trajectory. Usage is mandatory for multiple subject trials and optional for single subject trials.

4.12.32 SetStaticStatus

Declaration: HRESULT SetStaticStatus([in]BOOL i_IsStatic)

Return type: NONE

Parameters:

BOOL i_IsStatic	New static status flag

Description: Marks the trial as static or not. Static is used to indicate that the measurement subject remained still for the whole duration of the trial.

4.12.33 SetTrajectoryTypeGroupUnits

Declaration: HRESULT SetTrajectoryTypeGroupUnits([in]BSTR i_Name, [in]BSTR i_Units)

Return type: NONE

Parameters:

BSTR i_Name	Type group name
BSTR i_Units	Units

Description: Sets the units of the trajectory type group given by name. Units are normally the abbreviated SI form. See RegisterTrajectoryTypeGroup for more details.

4.12.34 Subject

Declaration: HRESULT Subject([in]long i_Index, [out, retval]Subject** retval)

Return type:

Subject	Subject interface

Parameters:

long i_Index	Zero-based subject index

Description: Obtains the specified subject as a pointer to a subject interface. Call Release on the object when it is no longer required.

If the subject index is out of range, the server throws an exception. Use SubjectCount to get the number of subjects present.

4.12.35 SubjectCount

Declaration: HRESULT SubjectCount([out, retval]long* retval)

Return type:	long	Subject count
Parameters:	NONE	

Description: Obtains the number of subjects associated with the trial.

4.12.36 Trajectory

Declaration: HRESULT Trajectory([in]long i_Index, [out, retval]Trajectory** retval)

Return type:	Trajectory	Trajectory interface
Parameters:	long i_Index	Zero-based trajectory index

Description: Obtains the specified trajectory as a pointer to a trajectory interface. Call Release on the object when it is no longer required.

If the trajectory index is out of range, the server throws an exception. Use TrajectoryCount to get the number of trajectories present.

4.12.37 TrajectoryCount

Declaration: HRESULT TrajectoryCount([out, retval]long* retval)

Return type:	long	Trajectory count
Parameters:	NONE	

Description: Obtains the number of 3-dimensional point trajectories in the trial.

4.12.38 TrajectoryTypeGroupCount

Declaration: HRESULT TrajectoryTypeGroupCount([out, retval]long* retval)

Return type:	long	Type group count
Parameters:	NONE	

Description: Obtains the number of defined type groups for trajectories. See RegisterTrajectoryTypeGroup for more details.

4.12.39 TrajectoryTypeGroupName

Declaration: HRESULT TrajectoryTypeGroupName([in]long i_Index, [out, retval]BSTR* retval)

Return type:	BSTR	Group name
Parameters:	long i_Index	Type group index

Description: Obtains the name of the trajectory type group given by index. See RegisterTrajectoryTypeGroup for more details.

4.12.40 TrajectoryTypeGroupUnits

Declaration: HRESULT TrajectoryTypeGroupUnits([in]BSTR i_Name, [out, retval]BSTR* retval)

Return type:	BSTR	Type group units
Parameters:	BSTR i_Name	Type group name

Description: Obtains the units of the trajectory type group given by name. Units are normally the abbreviated SI form. See RegisterTrajectoryTypeGroup for more details.

4.12.41 Type

Declaration: HRESULT Type([out, retval]BSTR* retval)

Return type:	BSTR	Type
Parameters:	NONE	

Description: Returns the trial type in the form of a text string.

4.12.42 UsesLabelPrefixes

Declaration: HRESULT UsesLabelPrefixes([out, retval]BOOL* retval)

Return type:	BOOL	Uses label prefix flag
Parameters:	NONE	

Description: Returns TRUE if label prefixes are in use in the trial, else FALSE.

4.12.43 VideoFieldCount

Declaration: HRESULT VideoFieldCount([out, retval]long* a_pCount)

Return type:	long	Field count
Parameters:	NONE	

Description: Obtains the number of fields allocated for the trial. This is normally the extent of the captured data but may be truncated. Processed 3D data may not necessarily exist for the full duration of the trial.

4.12.44 VideoRate

Declaration: HRESULT VideoRate([out, retval]double* rate)

Return type:	double	Video rate
Parameters:	NONE	

Description: Returns the sample rate of the trial video data in Hz.

5 ADDITIONAL UTILITY INTERFACES

5.1 Colour

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface provides a simple RGB colour implementation for setting and getting colours in other interfaces.
Methods:	
Properties:	1. Red 2. Green 3. Blue

5.1.1 Red

Declaration: [propput] HRESULT Red([in]short i_Value)
[propget] HRESULT Red([out, retval]short* retval)

Return type:	short	Red value
Parameters:	short i_Value	New red value

Description: Gets and sets the red component.

5.1.2 Green

Declaration: [propput] HRESULT Green([in]short i_Value)
[propget] HRESULT Green([out, retval]short* retval)

Return type:	short	Green value
Parameters:	short i_Value	New green value

Description: Gets and sets the green component.

5.1.3 Blue

Declaration: [propput] HRESULT Blue([in]short i_Value)
[propget] HRESULT Blue([out, retval]short* retval)

Return type:	short	Blue value
Parameters:	short i_Value	New blue value

Description: Gets and sets the blue component.

5.2 Point

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface provides a simple 3D point implementation for setting and getting 3D co-ordinates in other interfaces.
Methods:	1. IsValid
Properties:	1. X 2. Y 3. Z

5.2.1 IsValid

Declaration: HRESULT IsValid([out, retval]BOOL* retval)

Return type: BOOL — Valid flag for this point

Parameters: NONE

Description: Indicates if there is valid data for this point.

5.2.2 X

Declaration: [propput] HRESULT X([in]double i_Value)
[propget] HRESULT X([out, retval]double* retval)

Return type: double — X value

Parameters: double i_Value — New X value

Description: Sets and gets the X component.

5.2.3 Y

Declaration: [propput] HRESULT Y([in]double i_Value)
[propget] HRESULT Y([out, retval]double* retval)

Return type: double — Y value

Parameters: double i_Value — New Y value

Description: Sets and gets the Y component.

5.2.4 Z

Declaration: [propput] HRESULT Z([in]double i_Value)
[propget] HRESULT Z([out, retval]double* retval)

Return type: double — Z value

Parameters: double i_Value — New Z value

Description: Sets and gets the Z component.

5.3 Range

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface encapsulates a sample range. If the Size of the range is zero, then the range is not valid.
Methods:	1. Begin 2. End 3. Size
Properties:	

5.3.1 Begin

Declaration: HRESULT Begin([out, retval]long* retval)

Return type: long — The range start sample number

Parameters: NONE

Description: Gets the starting field number for the sample range.

5.3.2 End

Declaration: HRESULT End([out, retval]long* retval)

Return type: long — The range end sample number

Parameters: NONE

Description: Gets the ending field number for the sample range, which will be one less than Begin() + Size().

5.3.3 Size

Declaration: HRESULT Size([out, retval]long* retval)

Return type: long — The range length (in samples)

Parameters: NONE

Description: Returns the size of this range. A size of zero indicates that the range is invalid (or does not exist).

5.4 Ratio

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This is a simple helper class for wrapping up a fraction.
Methods:	1. Denominator 2. Numerator
Properties:	

5.4.1 Denominator

Declaration: HRESULT Denominator([out, retval]long* retval)

Return type: long Denominator value

Parameters: NONE

Description: Gets and sets the denominator of the ratio.

5.4.2 Numerator

Declaration: HRESULT Numerator([out, retval]long* retval)

Return type: long Numerator value

Parameters: NONE

Description: Gets and sets the numerator of the ratio.

5.5 Reaction

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface provides the user with information about force plate data.
Methods:	1. ForceX 2. ForceY 3. ForceZ 4. IsValid 5. MomentX 6. MomentY 7. MomentZ
Properties:	

5.5.1 ForceX

Declaration: HRESULT ForceX([out, retval]double* retval)

Return type: double — Force X value

Parameters: NONE

Description: Returns the X component of the force reaction.

5.5.2 ForceY

Declaration: HRESULT ForceY([out, retval]double* retval)

Return type: double — Force Y value

Parameters: NONE

Description: Returns the Y component of the force reaction.

5.5.3 ForceZ

Declaration: HRESULT ForceZ([out, retval]double* retval)

Return type: double — Force Z value

Parameters: NONE

Description: Returns the Z component of the force reaction.

5.5.4 IsValid

Declaration: HRESULT IsValid([out, retval]BOOL* retval)

Return type: BOOL — Valid flag for this point

Parameters: NONE

Description: Indicates if there is valid force and moment data.

5.5.5 MomentX

Declaration: HRESULT MomentX([out, retval]short* retval)

Return type: double — Moment X value

Parameters: NONE

Description: Returns the X component of the moment reaction.

5.5.6 MomentY

Declaration: HRESULT MomentY([out, retval]double* retval)

Return type: double — Moment Y value

Parameters: NONE

Description: Returns the Y component of the moment reaction.

5.5.7 MomentZ

Declaration: HRESULT MomentZ([out, retval]double* retval)

Return type: double — Moment Z value

Parameters: NONE

Description: Returns the Z component of the moment reaction.

5.6 Reconstruction

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface provides the user with detailed information on the properties of this reconstructed 3D point. Much of the functionality of this interface is also present in Trajectory.
Methods:	1. Camera 2. CameraCount 3. IsValid 4. Residual
Properties:	1. X 2. Y 3. Z

5.6.1 Camera

Declaration: HRESULT Camera([in]long i_Camera, [out, retval]long* retval)

Return type: long — The index of the camera

Parameters: NONE

Description: Obtains the one-based camera number of the Nth camera used to reconstruct the measured point, where N is given by the camera index. The total number of cameras contributing to the point can be obtained by calling CameraCount.

5.6.2 CameraCount

Declaration: HRESULT CameraCount([out, retval]long* retval)

Return type: long — Number of cameras contributing

Parameters: NONE

Description: Obtains the number of cameras that contributed to the reconstruction of the point. If the trajectory represents a point that was calculated rather than measured from a marker then the count will be zero.

5.6.3 IsValid

Declaration: HRESULT IsValid([out, retval]BOOL* retval)

Return type:	BOOL	Valid flag for this point
Parameters:	NONE	

Description: Indicates if there is valid data for this reconstruction.

5.6.4 Residual

Declaration: HRESULT Residual([out, retval]double* retval)

Return type:	double	Reconstruction residual
Parameters:	NONE	

Description: Returns the residual for reconstruction.

5.6.5 X

Declaration: [propput] HRESULT X([in]double i_Value)
[propget] HRESULT X([out, retval]double* retval)

Return type:	double	X value
Parameters:	double i_Value	New X value

Description: Returns the X component of the reconstruction.

5.6.6 Y

Declaration: [propput] HRESULT Y([in]double i_Value)
[propget] HRESULT Y([out, retval]double* retval)

Return type:	double	Y value
Parameters:	double i_Value	New Y value

Description: Returns the Y component of the reconstruction.

5.6.7 Z

Declaration: [propput] HRESULT Z([in]double i_Value)
[propget] HRESULT Z([out, retval]double* retval)

Return type:	double	Z value
Parameters:	double i_Value	New Z value

Description: Returns the Z component of the reconstruction.

5.7 RefCounter

Base interface:	IDispatch
Include file:	ServerPI.h
Typelib file:	PECS.tlb
Availability:	PECS 1.0, Workstation 4.5 with SDK 2.0
Description:	This interface is used internally by the PECS and the Plug-In to expose the server reference count (the number of attached client processes). Calling Decrement() and Increment() externally is not recommended and if the user wishes to know the application reference count, then PECS::ConnectionCount() is much more suitable. See the notes under PECS for a more detailed description of its use in Workstation Plug-Ins.
Methods:	1. Count 2. Decrement 3. Increment
Properties:	

5.7.1 Count

Declaration: HRESULT Count([out, retval]long* retval)

Return type: long — The application RefCount

Parameters: NONE

Description: Returns the reference count of this object (ordinarily, the number of client processes attached to the server).

5.7.2 Decrement

Declaration: HRESULT Decrement([out, retval]long* retval)

Return type: long — The new application RefCount

Parameters: NONE

Description: This function increments the reference count of this object only and returns the new reference count.

5.7.3 Increment

Declaration: HRESULT Increment([out, retval]long* retval)

Return type: long — The new application RefCount

Parameters: NONE

Description: This function decrements the reference count of this object only and returns the new reference count.